Best Local LLMs for Translation: What Actually Works

February 8, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** For high-resource language pairs (English-Spanish, English-French, English-German), Qwen 2.5 7B through Ollama gives you translation quality that rivals cloud services on just 5GB of VRAM. For low-resource languages or broad coverage (200+ languages), Meta's NLLB-200 is purpose-built for translation and runs the 1.3B model on 3GB of VRAM. For maximum speed on a single language pair, Opus-MT models are 300MB each and translate faster than any general LLM. Don't use a 32B general model for translation unless you also need it for other tasks — dedicated translation models are smaller, faster, and often better.

📚 **Related:** Qwen Models Guide · VRAM Requirements · Best LLMs for Chat · Voice Chat with Local LLMs

Translation is one of the strongest use cases for running AI locally. Your documents stay on your machine. No API costs per character. No rate limits when you're processing a thousand files. And unlike chat or coding, translation has dedicated small models that punch well above their weight — you don't need a 70B general model to get good results.

The catch: picking the right model depends entirely on which languages you need and what hardware you have. A 300MB Opus-MT model translates French-to-English faster and often better than a 14B parameter general LLM. But that same model can't handle Swahili, while NLLB-200 covers it natively.

Here's what actually works, what doesn't, and which model fits your setup.

Two Approaches: General LLMs vs Dedicated Translation Models

Before picking a model, understand the split.

General LLMs (Qwen, Llama, Mistral) can translate because they were trained on multilingual text. You prompt them with "Translate this to Spanish" and they do it. They're flexible — same model handles translation, summarization, chat, code. But they're large, and their translation quality varies wildly by language pair.

Dedicated translation models (NLLB, MADLAD-400, Opus-MT) were built specifically for translation. They're smaller, faster, and often better for the task — but translation is all they do. No chat, no summarization, no reasoning.

Approach	Strengths	Weaknesses
General LLM (Qwen, Llama)	Flexible, handles context/idiom, good for high-resource pairs	Large, slower, weak on rare languages
Dedicated (NLLB, Opus-MT)	Small, fast, broad language coverage, purpose-optimized	Translation only, less natural with idioms

If you're already running a general LLM for other tasks, try it for translation first. If you need dedicated translation infrastructure — batch processing, rare languages, maximum speed — use a purpose-built model.

General LLMs for Translation

Qwen 2.5: Best All-Around

Qwen 2.5 is the strongest general LLM for translation work. Alibaba trained it with explicit multilingual focus — 29 languages at the 7B size, with strong coverage across Chinese, English, and European languages.

Best for: Chinese-English (best in class at this size), European language pairs, any translation where you also want the model for other tasks.

Quality: On high-resource pairs like English-Spanish or English-German, Qwen 2.5 7B produces translations that read naturally — not the stilted output you'd expect from a 7B model. For Chinese-English specifically, it outperforms models twice its size.

Setup:

```
ollama run qwen2.5:7b
```

Then prompt with:

```
Translate the following text to Spanish. Output only the translation, no explanations.

[your text here]
```

VRAM: ~5GB at Q4 (7B), ~20GB at Q4 (32B for best quality)

Llama 3: Strong European, Weaker Asian

Llama 3's training data skewed heavily toward English and European languages. The result: excellent French, Spanish, German, Portuguese, and Italian translation. Weaker Japanese, Chinese, and Korean.

Best for: European language pairs where you want a non-Chinese-company model.

VRAM: ~5GB at Q4 (8B), ~20GB at Q4 (70B with multi-GPU)

Mistral/Mixtral: French Advantage

Mistral is a French company, and it shows. Mistral and Mixtral models handle French and European Romance languages particularly well. Mixtral 8x7B gives you strong multilingual performance if you have 24GB+ VRAM.

Best for: French-centric translation, European language pairs.

VRAM: ~5GB (Mistral 7B), ~26GB (Mixtral 8x7B at Q4)

Command-R: Built for Multilingual

Cohere's Command-R was specifically trained for multilingual use across 10 languages. It handles nuance and idiomatic expressions better than most open models at similar sizes.

Best for: When natural-sounding output matters more than raw accuracy (marketing copy, creative content).

VRAM: ~5GB at Q4 (35B needs ~20GB)

General LLM Translation Quality by Language Pair

Language Pair	Best General LLM	Quality vs Google Translate
English ↔ Chinese	Qwen 2.5 32B	Comparable or better
English ↔ Spanish	Qwen 2.5 7B / Llama 3 8B	Comparable

Language Pair	Best General LLM	Quality vs Google Translate
English ↔ French	Mistral 7B / Llama 3 8B	Comparable
English ↔ German	Qwen 2.5 7B / Llama 3 8B	Comparable
English ↔ Japanese	Qwen 2.5 32B	Slightly below
English ↔ Korean	Qwen 2.5 14B	Below
English ↔ Arabic	Qwen 2.5 14B	Below
English ↔ Swahili	Any general LLM	Significantly below — use NLLB

The pattern: high-resource European pairs work well even on 7B models. Asian languages need 14B+. Low-resource languages need dedicated models.

Dedicated Translation Models

NLLB-200: The Coverage King

Meta's No Language Left Behind covers 200 languages — including dozens that general LLMs barely handle. It improved translation quality by 44% average BLEU over previous state-of-the-art, with some African and Indian languages seeing 70%+ improvement.

Model sizes:

Model	Parameters	VRAM (FP16)	VRAM (INT8)	Quality
NLLB-200 Distilled 600M	600M	~2 GB	~1 GB	Good for common pairs
NLLB-200 Distilled 1.3B	1.3B	~3 GB	~2 GB	Sweet spot
NLLB-200 3.3B	3.3B	~8 GB	~4 GB	Best quality
NLLB-200 54.5B (MoE)	54.5B	~110 GB	~55 GB	Research only

Setup (Python + HuggingFace):

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

model_name = "facebook/nllb-200-distilled-1.3B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

```
# Translate English to French
tokenizer.src_lang = "eng_Latn"
inputs = tokenizer("Hello, how are you?", return_tensors="pt")
translated = model.generate(
    **inputs,
    forced_bos_token_id=tokenizer.convert_tokens_to_ids("fra_Latn")
)
print(tokenizer.batch_decode(translated, skip_special_tokens=True))
```

For faster inference, use CTranslate2 with INT8 quantization — 3-4x faster, half the memory:

```
pip install ctranslate2
# Pre-converted models available:
# OpenNMT/nllb-200-3.3B-ct2-int8
```

Best for: Low-resource languages, broad coverage, batch processing, privacy-critical workflows.

MADLAD-400: Broadest Coverage

Google's MADLAD-400 supports 419 languages — more than double NLLB's coverage. Quality is comparable to NLLB on common pairs but the real value is those extra 200+ languages.

Model sizes: 3B, 7.2B, 10.7B

Setup:

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

model = AutoModelForSeq2SeqLM.from_pretrained("google/madlad400-3b-mt")
tokenizer = AutoTokenizer.from_pretrained("google/madlad400-3b-mt")

# Prefix with target language code
inputs = tokenizer("<2pt> I love pizza!", return_tensors="pt")
outputs = model.generate(**inputs)
print(tokenizer.batch_decode(outputs, skip_special_tokens=True))
# Output: "Eu amo pizza!"
```

Best for: Extremely rare languages not covered by NLLB, research applications.

Trade-off vs NLLB: Broader coverage but slightly lower quality on common pairs. NLLB 3.3B generally outperforms MADLAD-400 3B on well-resourced language pairs.

Opus-MT: Lightweight Specialists

Helsinki-NLP's Opus-MT takes the opposite approach: instead of one model for all languages, it provides 1,200+ individual models — one per language direction. Each model is roughly 300MB.

Why this matters: If you only need English-to-Spanish translation, why load a 3GB model that handles 199 other languages you'll never use?

Setup:

```
from transformers import pipeline

translator = pipeline("translation", model="Helsinki-NLP/opus-mt-en-es")
result = translator("The weather is nice today.")
print(result[0]['translation_text'])
```

Best for: Single language pairs, embedded/edge devices, maximum speed, minimal resource usage.

Limitation: Each model handles one direction. English→Spanish and Spanish→English are separate models. For bidirectional work across multiple languages, NLLB is more practical.

Seamless M4T: When You Need Speech Too

Meta's Seamless M4T v2 handles text-to-text, speech-to-text, text-to-speech, and speech-to-speech translation in a single model. If your workflow involves audio — translating podcasts, meetings, video content — this is the only local option that does it all.

Model sizes: 281M (on-device), 1.2B (medium), 2.3B (large, ~6GB VRAM)

Language coverage: ~100 languages for speech, 200 for text.

Quality: 20% BLEU improvement over previous speech translation models. Text-to-text quality matches NLLB 3.3B.

Best for: Workflows that mix speech and text translation. If you only need text, NLLB is simpler.

Quality Reality Check

How do local models actually compare to cloud translation services?

High-Resource Pairs (EN-ES, EN-FR, EN-DE)

Local quality: Excellent. Qwen 2.5 7B and NLLB 3.3B both produce translations that are usable without editing for most purposes. Professional translators would still find errors, but for business correspondence, documentation, and casual use, local models match Google Translate.

Where cloud still wins: Highly idiomatic text, marketing copy, literary translation. DeepL in particular handles tone and register better than any local model.

Medium-Resource Pairs (EN-JA, EN-ZH, EN-KO)

Local quality: Good but imperfect. Qwen 2.5 handles Chinese-English at near-cloud quality. Japanese and Korean are weaker — grammar structures that differ fundamentally from English trip up smaller models. A 32B model handles these noticeably better than a 7B.

Where cloud still wins: Honorifics, formality levels, cultural context. Japanese keigo (politeness levels) is particularly hard for local models under 32B.

Low-Resource Pairs (EN-SW, EN-TL, EN-AM)

Local quality: This is where dedicated models justify their existence. NLLB 1.3B translates Swahili, Tagalog, and Amharic at quality levels that general LLMs can't touch — because general LLMs barely saw these languages in training.

Cloud comparison: Google Translate covers 249 languages but quality for low-resource pairs varies. NLLB often matches or beats Google on languages like Yoruba, Igbo, and Hausa where Meta specifically invested in training data.

The Honest Summary

Language Tier	Best Local Option	vs Cloud Services
High-resource European	Qwen 2.5 7B or Llama 3 8B	90-95% of cloud quality
Chinese-English	Qwen 2.5 14B+	95%+ of cloud quality
Japanese/Korean	Qwen 2.5 32B	80-90% of cloud quality

Language Tier	Best Local Option	vs Cloud Services
Low-resource African/Asian	NLLB 3.3B	Matches or beats Google
Rare languages (400+)	MADLAD-400	Often the only option

VRAM Requirements Summary

Model	VRAM Needed	Languages	Best For
Opus-MT (per direction)	~300 MB	1 pair per model	Single-pair, max speed
NLLB 600M (distilled)	~2 GB	200	Low-resource, tight hardware
NLLB 1.3B (distilled)	~3 GB	200	Best quality/size ratio
Seamless M4T Large	~6 GB	100 (speech) / 200 (text)	Speech + text translation
Qwen 2.5 7B (Q4)	~5 GB	29	General-purpose + translation
NLLB 3.3B	~8 GB	200	Best dedicated quality
MADLAD-400 3B	~7 GB	419	Broadest coverage
Qwen 2.5 14B (Q4)	~9 GB	29	Better Asian language quality
Qwen 2.5 32B (Q4)	~20 GB	29	Best general LLM translation

If you have 8GB of VRAM or less, dedicated translation models are your best option. You can run NLLB 1.3B and get 200-language coverage on a GTX 1060. A general LLM at that VRAM tier (7B at Q4) translates well for common pairs but can't touch NLLB's language breadth.

→ Use our Planning Tool to check exact VRAM for your setup.

With 24GB of VRAM, you can run Qwen 2.5 32B for the highest-quality general translation, or NLLB 3.3B alongside other models for dedicated translation tasks.

Practical Setups

Simple Translation: Ollama + Qwen 2.5

The fastest way to start translating locally. Works for high-resource language pairs.

```
# Install Ollama
curl -fsSL https://ollama.com/install.sh | sh

# Pull Qwen 2.5 7B
ollama pull qwen2.5:7b

# Translate via API
curl http://localhost:11434/api/generate -d '{
  "model": "qwen2.5:7b",
  "prompt": "Translate to German. Output only the translation:\n\nThe project deadline has been
  "stream": false
}'
```

Pros: Zero setup beyond Ollama. Same model handles translation and everything else. **Cons:** Slower than dedicated models. Weaker on rare languages.

Production Quality: NLLB via HuggingFace

For batch translation or language pairs where quality matters.

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
import torch

model_name = "facebook/nllb-200-3.3B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16
).cuda()

def translate(text, src_lang, tgt_lang):
    tokenizer.src_lang = src_lang
    inputs = tokenizer(text, return_tensors="pt").to("cuda")
    translated = model.generate(
        **inputs,
        forced_bos_token_id=tokenizer.convert_tokens_to_ids(tgt_lang),
        max_length=512
    )
    return tokenizer.batch_decode(translated, skip_special_tokens=True)[0]

# English to Swahili
print(translate("Hello, how are you?", "eng_Latn", "swh_Latn"))
```

```
# French to Japanese
print(translate("Bonjour le monde", "fra_Latn", "jpn_Jpan"))
```

Pros: 200 languages, optimized for translation, smaller than general LLMs. **Cons:** Python required. No chat capability.

Document Translation: Chunk and Reassemble

For translating long documents, you need to chunk the text to stay within context limits while keeping paragraph structure intact.

```
def translate_document(text, src_lang, tgt_lang, max_chunk=400):
    paragraphs = text.split('\n\n')
    translated_paragraphs = []

    for paragraph in paragraphs:
        if len(paragraph.split()) <= max_chunk:
            translated_paragraphs.append(
                translate(paragraph, src_lang, tgt_lang)
            )
        else:
            # Split long paragraphs at sentence boundaries
            sentences = paragraph.split('. ')
            chunk = []
            for sentence in sentences:
                chunk.append(sentence)
                if len(' '.join(chunk).split()) > max_chunk:
                    translated_paragraphs.append(
                        translate('. '.join(chunk[:-1]) + '.', src_lang, tgt_lang)
                    )
                    chunk = [sentence]
            if chunk:
                translated_paragraphs.append(
                    translate('. '.join(chunk), src_lang, tgt_lang)
                )

    return '\n\n'.join(translated_paragraphs)
```

Key gotcha: Don't split mid-sentence. Translation models need sentence-level context to produce correct grammar. Splitting at word boundaries produces garbage.

When to Use Cloud Instead

Local translation isn't always the right call.

Legal or medical documents: Liability matters. Professional translation services provide certified translations with accountability. Running a legal contract through NLLB and calling it translated is asking for trouble.

50+ language pairs with high quality: If you need production-quality translation across dozens of language pairs simultaneously, DeepL or Google Translate's API is more practical than managing multiple local models.

Real-time speech translation: Seamless M4T handles this locally, but cloud services (Google, Azure) are more polished for real-time bidirectional speech translation in production.

One-off translations: If you translate three sentences a month, installing models and writing Python scripts is overkill. Just use DeepL.

The break-even point: If you translate more than ~100,000 characters per month, local models save money over API pricing. Below that, the setup time isn't worth it unless privacy is the priority.

The Bottom Line

Local translation is more practical than most people expect. You don't need a 70B model or a 3090 — a $200 GPU running NLLB 1.3B handles 200 languages at quality that matches cloud services for most pairs.

Start here:

- **Common European pairs:** Ollama + Qwen 2.5 7B (5GB VRAM, zero setup)
- **Broad language coverage:** NLLB 1.3B distilled (3GB VRAM, 200 languages)
- **Single language pair, max speed:** Opus-MT (300MB, fastest option)
- **Speech + text:** Seamless M4T v2 (6GB VRAM)
- **Best possible quality:** Qwen 2.5 32B (20GB VRAM) for common pairs, NLLB 3.3B (8GB VRAM) for everything else

The models that surprise people most: NLLB at 1.3B parameters translating Yoruba better than a 14B general model, and Opus-MT at 300MB translating French faster than anything ten times its size. Bigger isn't always better — especially for translation.

📚 **More on this topic:** Qwen Models Guide · VRAM Requirements · Best LLMs for Chat · Best Models for Summarization

Get notified when we publish new guides.

Subscribe — free, no spam

Source: https://insiderllm.com/guides/best-local-llms-translation/