MASTER'S THESIS 2026

Breaking Language Barriers in Real Time: On-Device AI-Driven Speech Translation

Adrian Steene, Lucas Wittich

ISSN 1650-2884
LU-CS-EX: 2026-51

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-51

Breaking Language Barriers in Real Time: On-Device AI-Driven Speech Translation

Bryta Språkbarriärer i Realtid: Enhetsbaserad AI-driven Talöversättning

Adrian Steene, Lucas Wittich

Breaking Language Barriers in Real Time: On-Device AI-Driven Speech Translation

Adrian Steene
ad3122st@student.lu.se

Lucas Wittich
lu1842wi@student.lu.se

July 15, 2026

Master's thesis work carried out at Axis Communications AB.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Zebastian Karra-Krüger, zebastian.karra-kruger@axis.com
Dan Lundgren, dan.lundgren@axis.com

Examiner: Xuan-Son Vu, xuan-son.vu@cs.lth.se

Abstract

This study investigates whether multilingual speech translation can run locally under real-time and resource limits. We implement a cascaded pipeline where automatic speech recognition transcribes source-language speech and machine translation generates target-language text. The system is evaluated across six European languages through component-level and end-to-end experiments.

The ASR evaluation compares multilingual recognition models and further examines the strongest candidate using ONNX Runtime execution and dynamic INT8 quantization. The MT study compares multilingual translation models and investigates LoRA-based fine-tuning for weak language pairs. The components are then integrated and evaluated as a complete speech translation pipeline.

The results indicate that accurate real-time ASR is feasible under an 8 GiB, four-core setting after deployment-oriented optimisation. However, the full pipeline is mainly limited by MT latency, and translation quality degrades when MT receives ASR-generated transcripts. Local multilingual speech translation is therefore technically feasible within the evaluated setup, but not yet production-ready.

Keywords: On-Device AI, Speech Translation, Automatic Speech Recognition, Machine Translation, Resource-constrained Inference, Quantization, LoRA Fine-tuning

Acknowledgements

We would like to express our sincere gratitude to Pierre Nugues for his guidance, unwavering engagement, and constructive feedback throughout this process.

We also thank Axis Communications for providing the opportunity to carry out this thesis. During the project, we had the privilege of meeting many colleagues whose experience, curiosity, and perspectives helped shape our understanding of practical software engineering, localisation, and deployment challenges. Their openness and inspiration were valuable throughout the process.

Finally, we thank our families and friends for their encouragement, patience, and support throughout our studies. We are especially grateful to Viktor Månsson and Christoffer Sylve for their close friendship and for the many moments shared during the past five years at Lund University.

Acronyms

ASR automatic speech recognition.

BLEU Bilingual Evaluation Understudy.

LLM large language model.

LoRA Low-Rank Adaptation.

MoE Mixture of Experts.

MT machine translation.

NLLB No Language Left Behind.

NMT neural machine translation.

ONNX Open Neural Network Exchange.

PEFT parameter-efficient fine-tuning.

PTQ post-training quantization.

RSS resident set size.

RTF real-time factor.

TDT Token-and-Duration Transducer.

TPS tokens per second.

WER word error rate.

Contents

Chapter 1
Introduction

1.1 Context and Motivation

Network-connected products are increasingly deployed in environments where users, visitors, and operators do not necessarily share a common language. Localisation in these contexts is not limited to menus, documentation, or static interface text (Esselink, 2000). When communication must happen immediately, spoken interaction can also become part of the user experience.

For a speech-driven translation system, translation quality is only one part of the problem. The system must also respond quickly enough to support an ongoing interaction and run within the CPU and memory budget of the device that captures the speech. Local multilingual speech translation is therefore a system-level problem, where transcription quality, translation quality, latency, CPU utilisation, and memory consumption must be evaluated together.

Cloud-based speech and translation services can provide access to large models and centralised infrastructure, but they also introduce external dependencies. Network availability, bandwidth, service latency, and remote data processing can all affect whether the system is actually usable in practice (Shi et al., 2016; Mao et al., 2017). Speech data may also contain personal or otherwise sensitive information, both through the spoken content and speaker-related characteristics (European Data Protection Board, 2021). Edge computing addresses part of this problem by moving computation closer to the data source, but it also shifts the burden to the device itself, where CPU capacity, memory, power, and runtime behaviour are limited (Shi et al., 2016; Xu et al., 2021).

We study this trade-off through a cascaded multilingual speech translation pipeline. An automatic speech recognition (ASR) model converts source-language speech into text, and a machine translation (MT) model translates the transcript into the target language (Sperber and Paulik, 2020). This modular structure allows the ASR and MT components to be selected, evaluated, and optimised independently. However, it also introduces two important

risks. Recognition errors may propagate into the translation stage and degrade translation quality (Khayrallah and Koehn, 2018; Sperber and Paulik, 2020), while latency accumulates across the sequential components. Feasibility must therefore be evaluated over the complete ASR-MT pipeline, not only through isolated component scores.

1.2 Problem Statement

Axis Communications develops network-based surveillance and security products, including cameras, audio devices, and intercoms. In such systems, multilingual spoken interaction can be valuable when users do not share a common language. The deployment challenge is that speech recognition and translation must be performed under the limited computational and memory resources available on the device while still producing output quickly enough for interactive use.

Speech translation services are commonly deployed through remote infrastructure, where model execution takes place on external servers. This can simplify model management and provide access to larger models, but it also introduces dependencies on connectivity, remote service latency, and external data processing. In this work, we instead evaluate local inference. This does not by itself resolve all privacy or deployment concerns, but it can reduce reliance on continuous connectivity and external data processing.

The central problem is whether a cascaded ASR-MT pipeline can be executed locally under constrained CPU and memory conditions while preserving sufficient transcription quality, translation quality, and end-to-end latency for interactive speech translation. We treat this as a deployment feasibility problem rather than introducing new model architectures. The evaluation uses existing pretrained ASR and MT models, applies deployment-oriented optimisation where appropriate, and measures the resulting trade-offs between accuracy, latency, CPU utilisation, and memory consumption.

1.3 Objectives

The evaluation is guided by the following objectives:

1. Evaluate whether the complete ASR-MT pipeline can operate at or below real time under the evaluated resource limits.

2. Characterise the trade-offs between transcription quality, translation quality, latency, CPU use, and memory consumption.

3. Assess how model selection, deployment-oriented ASR optimisation, and MT fine-tuning affect system-level feasibility.

4. Compare clean reference-input translation with noisy ASR-input translation to estimate how recognition errors affect downstream MT quality.

1.4 Contributions

The work makes the following contributions:

- We design and implement a cascaded multilingual ASR-MT pipeline for local speech translation under deployment-oriented CPU and memory limits.
- We construct a six-language evaluation corpus derived from FLEURS, using parallel sample identifiers to support consistent ASR, MT, and end-to-end evaluation across English, Swedish, Dutch, French, German, and Spanish.
- We evaluate multilingual ASR models under both accuracy-oriented and resource-constrained settings, measuring WER, latency, real-time factor, CPU use, and memory consumption.
- We evaluate multilingual MT models for the selected language set and investigate LoRA-based fine-tuning strategies for weak translation directions.
- We conduct a focused deployment study of the strongest ASR candidate, evaluating ONNX Runtime execution, exported graph preparation, and dynamic INT8 quantization scope.
- We evaluate the integrated cascade under clean reference-input translation and noisy ASR-input translation to quantify the effect of recognition errors on downstream translation quality.

1.5 Summary of Main Findings

The results show a conditional feasibility result. Under the evaluated 8 GiB, four-core setting, the ASR component can meet real-time requirements when the Parakeet ASR system is executed through ONNX Runtime and selectively quantized with dynamic INT8 MatMul and Gemm quantization. In that configuration, ASR is accurate, fast, and no longer the main latency bottleneck.

The complete ASR-MT pipeline remains more limited. End-to-end latency is dominated by the MT component, and the 95th percentile pipeline RTF exceeds the real-time threshold. Translation quality also decreases when the MT model receives ASR-generated transcripts rather than clean reference text. The system should therefore be interpreted as a technical feasibility prototype rather than a production-ready solution.

1.6 Scope and Delimitations

This study focuses on the feasibility of local multilingual speech-to-text translation using a cascaded ASR-MT pipeline. Text-to-speech synthesis, speech-to-speech translation, user-interface integration, and user satisfaction studies are outside the empirical scope.

The study is limited to six European languages: English, Swedish, Dutch, French, German, and Spanish. These languages were selected because they are supported by the evaluated

models, available in the evaluation datasets, and sufficiently familiar to us to support qualitative inspection of model outputs. The results should not be interpreted as evidence of general multilingual performance across all languages or language families.

The experiments are deployment-oriented, but they are not performed on final embedded hardware. Runtime experiments are conducted in constrained containerised environments with controlled CPU and memory limits. These limits allow comparison between model configurations, but they do not capture all properties of a physical embedded device, such as thermal behaviour, accelerator availability, power consumption, or real-time operating-system scheduling.

Speaker diarization is relevant for practical multi-speaker deployment because a complete spoken interaction system may need to identify speaker turns and preserve conversational structure, but we do not evaluate diarization experimentally (Park et al., 2021). All reported results assume utterance-level input without speaker attribution, and the conclusions concern local ASR-MT feasibility rather than complete multi-speaker conversation handling.

This work does not propose new ASR or MT architectures. Instead, it evaluates existing pretrained models and investigates whether model selection, MT fine-tuning, ASR backend selection, and ASR quantization can make a cascaded pipeline feasible under local inference constraints. Privacy and data-protection considerations motivate local processing, but we do not perform a formal privacy, security or legal compliance analysis.

1.7 Division of Work

The project was conducted jointly, with shared responsibility for system design, experimental planning, integrated pipeline evaluation, and interpretation of the final results. The component-level work was divided between us. Adrian Steene was primarily responsible for the MT experiments, MT fine-tuning, and construction of the fine-tuning data pipeline. Lucas Wittich was primarily responsible for the ASR experiments, deployment-oriented benchmarking, runtime measurement framework, and the focused ASR backend and quantization study.

Although the component-level experiments were divided between us, the scientific claim concerns the combined system. The final conclusions therefore address the behaviour of the complete pipeline under local deployment constraints rather than the isolated contribution of either component.

1.8 Thesis Outline

Chapter 2 reviews related work on cascaded speech translation, multilingual ASR and MT, efficient on-device inference, and model optimisation. Chapter 3 introduces the technical background needed to interpret the experiments, including cascaded ASR and MT systems, error propagation, latency accumulation, quantization, fine-tuning, and evaluation metrics. Chapter 4 describes the FLEURS-derived evaluation dataset and the TED2020-derived fine-tuning corpus. Chapter 5 presents the methodology, including the runtime environment, preprocessing, model selection, measurement procedures, and integrated pipeline evaluation. Chapter 6 reports and interprets the component-level and end-to-end results. Chapter 7

summarises the main findings, states the conditional feasibility claim, and outlines limitations and future work.

Together, these chapters frame local multilingual speech translation as a combined quality, latency, and resource-use problem. The central question is not whether ASR or MT can perform well in isolation, but whether current components can be combined into a local pipeline that remains useful under the evaluated deployment limits.

Chapter 2

Related Work

Local speech translation sits at the intersection of speech translation, multilingual ASR, multilingual MT, and efficient neural inference. Prior work provides strong components in each area, but component quality alone does not answer our deployment question. A local speech-driven translation system must recognise speech, translate the recognised text, and do so within explicit latency, CPU, and memory limits. This chapter therefore focuses on work that explains the main design trade-offs behind the evaluated pipeline and motivates the gap addressed by the evaluation.

2.1 Speech Translation and Localisation

Localisation traditionally concerns the adaptation of software, documentation, and user-facing content to different languages and cultural contexts (Esselink, 2000). In products that support spoken interaction, localisation also becomes a speech and language-processing problem. A system that receives spoken input must recognise the content, convert it into the relevant target language, and do so quickly enough to remain useful during the interaction (Sperber and Paulik, 2020).

Edge computing provides an interesting deployment perspective for this problem. Moving computation closer to the data source can reduce reliance on continuous connectivity and remote service availability, but it also shifts the computational burden to the device (Shi et al., 2016; Mao et al., 2017; Xu et al., 2021). For speech translation, this trade-off is especially important because the input is time-dependent and may contain sensitive content (European Data Protection Board, 2021). A local speech translation system must therefore be evaluated not only by translation quality, but also by latency, memory use, CPU utilisation, and the practical cost of running the models in the target environment.

2.2 Cascaded Speech Translation

Speech translation systems are commonly set up as either direct speech-to-text translation models or as cascaded systems in which ASR first produces a source-language transcript and MT then translates that transcript into the target language (Sperber and Paulik, 2020). The cascaded design remains attractive because it allows ASR and MT components to be reused, evaluated, and improved independently. This modularity is especially useful in deployment-oriented work because the two components may differ in model size, runtime behaviour, optimisation options, and failure modes.

The same modularity also introduces two limitations that are central to this study. First, errors from the ASR stage become input noise for the MT stage. Prior work on noisy input in neural machine translation (NMT) shows that substitutions, deletions, insertions, and other source-side disturbances can degrade translation quality (Khayrallah and Koehn, 2018). In a cascaded speech translation system, transcription quality is therefore not only an ASR metric as it also affects the final translated output. Second, latency accumulates across the sequential stages. A component that performs well in isolation may therefore still be unsuitable if it causes the full pipeline to miss real-time requirements.

Recent work on low-latency cascaded speech translation addresses this problem by allowing the translation model to consume partial hypotheses rather than waiting for complete transcription. Alignment-based streaming MT approaches introduce adaptive read/write policies that decide when sufficient source context is available to emit target-language output (Ahmed et al., 2025). Such work shows that latency in cascaded speech translation is shaped not only by model speed, but also by how the MT stage consumes upstream hypotheses. We do not implement these policies, we instead evaluate a strictly sequential pipeline to establish a local baseline and identify the main bottlenecks before adding streaming-specific complexity.

2.3 ASR for Multilingual Speech Translation

ASR forms the upstream component of a cascaded speech translation system, so its suitability depends on more than WER. The recogniser must support the selected languages, produce transcripts that remain useful for MT, and run within the available latency and memory budget. Earlier large-vocabulary ASR systems were commonly based on hybrid HMM-GMM pipelines, where acoustic, pronunciation, and language models were developed as separate components (Rabiner, 1989; Jurafsky and Martin, 2026). Modern ASR research has largely moved towards end-to-end neural architectures that learn the mapping from acoustic input to textual output within a single model (Prabhavalkar et al., 2024).

Two ASR model families are evaluated in this study. The first consists of multilingual encoder-decoder models from the whisper family. Whisper is trained on large-scale multilingual and multitask speech data and supports transcription, language identification, and speech translation into English within the same framework (Radford et al., 2022). Its broad language coverage makes it a strong candidate for this evaluation. However, its encoder-decoder structure relies on autoregressive decoding and is primarily used in full-sequence or chunk-based settings. This makes Whisper useful as a multilingual accuracy baseline, but its latency and memory behaviour must be measured before it can be considered suitable for

local real-time deployment.

The second family consists of transducer-based ASR models, which are designed for incremental token emission and are therefore more closely aligned with low-latency speech processing. Parakeet-TDT-0.6B, developed by NVIDIA and released under the CC-BY-4.0 licence, belongs to this family (Sekoyan et al., 2025; Creative Commons, 2021a). It covers 25 different European languages and combines a FastConformer encoder with a Token-and-Duration Transducer decoder, which is further described in Section 3.3.2 (Sekoyan et al., 2025; Rekesh et al., 2023; Xu et al., 2023). This architecture is relevant because it targets efficient speech recognition while maintaining strong multilingual performance. In this study, Parakeet is not just another ASR model. It is also used to test whether a larger and more accurate recogniser can still satisfy local real-time constraints when combined with an appropriate runtime backend and numerical optimisation.

2.4 Efficient On-Device Inference

Deploying modern neural models on edge and embedded hardware is limited by memory capacity, memory bandwidth, and computational resources. Prior work on edge computing argues that moving computation closer to the data source can reduce communication dependency and improve local autonomy, but it also transfers computational burden to the device (Xu et al., 2021; Shi et al., 2016). For neural speech and translation models, this makes deployment feasibility a delicate balance between the model, inference backend, and hardware environment.

Model compression and runtime optimisation are two common approaches to handle these limits. Quantization is a technique that reduces the numerical precision of weights, and, in some cases activations, in order to reduce storage requirements and improve inference efficiency. Post-training quantization is particularly relevant for deployment because it can be applied without retraining the full model (Nagel et al., 2021; Gholami et al., 2021). However, the benefit of quantization is not automatic. Accuracy, latency, and memory behaviour depend on the quantization scheme, the way activation ranges are estimated, and the availability of low-precision kernels.

Transformer-oriented quantization work illustrates both the potential and limits of quantizing. Dettmers et al. (2022) show that large Transformer models can be quantized to 8-bit matrix multiplication while preserving performance by treating outlier features separately. Activation-aware weight quantization similarly demonstrates that low-bit weight-only quantization can reduce memory requirements for large language models while retaining much of the original model quality (Lin et al., 2024). These studies motivate quantization as a deployment strategy, but also show why its effect must be measured. In this work we focus on operator-level quantization, in order to understand what quantization configuration gives the best balance between accuracy, latency, and hardware use.

Another factor that can affect deployment is the choice of inference backend. Open Neural Network Exchange (ONNX) Runtime provides graph-level optimisation and quantization workflows for executing exported models across different environments (ONNX Runtime, 2025). The same pretrained ASR model may therefore show different latency and memory behaviour depending on whether it is executed through its original framework or through an exported graph. This makes backend selection part of the empirical feasibility question

rather than purely implementation-level detail.

2.5 Multilingual Machine Translation

The MT component in the pipeline determines whether the source-language transcript can be converted into useful target-language text. Prior work in multilingual MT highlights different strategies for balancing language coverage, model specialization, and deployment cost. Some systems prioritize compact, direction-specific translation models that are easy to deploy, while others aim to support many language pairs within a single shared multilingual model. Within on-device speech translation this balance becomes challenging as multilingual support may reduce system complexity at the model-management level, but can also increase memory usage and inference cost.

2.5.1 OPUS-MT

OPUS-MT is a collection of open source NMT models and tools developed by the University of Helsinki as part of the OPUS project released under the CC-BY-4.0 licence (Tiedemann and Thottingal, 2020; Creative Commons, 2021a). Rather than building a single universal model, the project provides a large repository of pre-trained translation models for specific language directions implementing the transformer architecture. These OPUS-MT models are particularly relevant because they represent the opposite design choice from a single broad multilingual model. These compact, direction-specific models can be easier to run individually, but supporting many language pairs requires managing several models.

For constrained systems, this creates a practical trade-off. A task-specific model may be moderate in size, but broad multilingual support can require multiple models to be stored, loaded, and selected at runtime. For a speech translation system intended to support several languages on-device, model-management complexity and aggregate storage can therefore become as important as the size of any single model.

2.5.2 mBART-50

The mBART-50-many-to-many model extends the original mBART idea of multilingual denoising sequence-to-sequence pretraining to 50 languages and is intended to support multilingual transfer for generation tasks, among them translation (Tang et al., 2020). It is trained on large-scale monolingual corpora and subsequently fine-tuned for machine translation tasks. By introducing language-specific tokens and a shared encoder-decoder architecture, mBART-50 supports translation between arbitrary language pairs without requiring separate models. This approach enables effective cross-lingual transfer, particularly for low-resource languages, although performance may vary depending on language similarity and training data distribution.

2.5.3 M2M-100

M2M-100 is a multilingual MT model designed to support direct many-to-many translation across 100 languages without relying on English as a pivot language (Fan et al., 2020). By

training a single Transformer-based encoder-decoder model on parallel data covering multiple language pairs, the model does not need to translate with intermediate steps, thereby reducing error propagation and improving translation efficiency. This approach represents a significant shift from traditional pivot-based systems, although performance may still vary depending on data availability for specific language pairs.

2.5.4 NLLB-200

The No Language Left Behind (NLLB) model, developed by Meta and released under the CC-BY-NC-4.0 licence, significantly extends multilingual translation coverage to over 200 languages (Team et al., 2022; Creative Commons, 2021b). The model emphasizes translation between low-resource languages and is trained on a large-scale corpus constructed using automatic data mining and filtering techniques. NLLB is built on the encoder-decoder transformer architecture and is released in several variants:

- A dense transformer available with 1.3B and 3.3B parameters.
- A Sparsely Gated Mixture of Experts (MoE) model with 54.5B parameters.
- Distilled models with 600M and 1.3B parameters, obtained via knowledge distillation from the MoE model. The distilled 600M variant is the focus of this thesis.

One of the main innovations in NLLB is the use of MoE. MoE replaces the feed-forward network (FFN) sublayers in both the encoder and decoder. Utilizing MoE allows the model to improve performance while maintaining training and inference FLOPs comparable to much smaller dense models. MoE works by routing tokens to specific parts of the model. This means that only a fraction of the 54.5B parameters are active for a given input. In the paper, the MoE model achieves the highest Bilingual Evaluation Understudy (BLEU) scores among the evaluated variants. When comparing the distilled 600M model with the dense 600M model, the distilled performs as well or better for all tested languages even if the results are quite similar.

2.6 Large Language Models for Translation

Recent research has also explored translation with large language models, where translation is framed as a conditional text generation task by a general-purpose multilingual model rather than a dedicated NMT system (Brown et al., 2020; Zhao et al., 2025). Such models are useful because they can support multilingual, zero-shot, and few-shot translation capabilities within a single framework. A recent example is TranslateGemma, Googles translation focused model built on top of the Gemma 3 foundation model (Google Translate Research Team et al., 2026). Unlike general-purpose multilingual language models, TranslateGemma is specifically optimized for machine translation through translation focused fine-tuning and reinforcement learning. The model family is evaluated across 55 language pairs and is available in several sizes, with the smallest variant containing 4B parameters. Similarly, the Qwen family has introduced Qwen3-MT, a dedicated translation model built on the Qwen3 foundation model, that extends support to 92 languages while achieving high translation quality (Yang et al., 2025; Team, 2025).

We exclude LLM-based translation from the main evaluation because the study focuses on local real-time feasibility under constrained CPU and memory budgets. Large autoregressive models can impose substantial memory and decoding costs, making them difficult to compare fairly within the same deployment envelope as the ASR-MT pipeline evaluated here (Wan et al., 2024). Conventional multilingual NMT models therefore provide a more controlled baseline for studying the interaction between translation quality, latency, and local resource use. LLM-based speech translation remains relevant for future work, especially if smaller models, hardware acceleration, or more efficient decoding methods become available in the target deployment environment.

2.7 Summary and Identified Research Gap

Prior work provides strong individual components for multilingual ASR, multilingual MT, low-latency speech processing, and efficient neural inference. What remains less clear is whether these components can be combined into a local multilingual speech translation system that satisfies quality, latency, and memory limits at the same time. ASR studies often emphasise transcription quality or streaming behaviour, while MT studies usually evaluate translation quality on clean text input. Efficient inference work often studies compression or runtime acceleration in isolation. In an on-device speech translation system, these dimensions interact. ASR errors affect MT quality, ASR and MT latency accumulate, and memory limits can exclude otherwise accurate models. Table 2.1 outlines where this thesis sits in relation to other popular speech translation frameworks across these dimensions.

Work	Architecture	Local	Multilingual	Latency	Support Constraint Resources
Seamless Communication et al. (2023)	End-to-End	No	Yes	Yes	No
Ahmed et al. (2025)	Cascaded S2TT with streaming MT	Yes	No	Yes	No
Gemini 3.5 Live Translate (Google DeepMind, 2026)	End-to-End Multimodal	No	Yes	Yes	No
This thesis	**Cascaded S2TT**	**Yes**	**Yes**	**Yes**	**Yes**

Table 2.1: Comparison of representative speech translation systems and this thesis.

As seen in Table 2.1, existing speech translation frameworks compromise on either multilingual translation capabilities or model size. While powerful multimodal models like Seamless (Seamless Communication et al., 2023) and Gemini 3.5 Live Translate (Google DeepMind, 2026) support multilingual, low-latency translation, they are very computationally demanding. This means that they are not feasible for on-device deployment. Meanwhile, models like Ahmed et al. (2025) achieve on-device capabilities but rely on isolated bilingual translation models. This study addresses that gap by evaluating existing ASR and MT models as parts of a local cascaded pipeline under controlled CPU and memory constraints. The evaluation measures the resulting trade-offs between transcription quality, translation quality, latency, CPU utilisation, and memory constraints. We also examine whether deployment-oriented ASR optimisation through backend selection and dynamic INT8 quantization can improve

feasibility without unacceptable recognition degradation. The aim is not to propose an entirely new speech translation architecture, but to test whether current components can satisfy the combined requirements of local multilingual speech translation in the evaluated setting.

Chapter 3
Background

Local speech translation combines language modelling with a deployment problem. The system must recognise speech, translate the recognised text, and return the output quickly enough for interaction while staying within local CPU and memory limits. This chapter introduces the concepts needed to interpret the experiments. It explains how errors and latency propagate through a cascaded ASR-MT pipeline, why the evaluated ASR and MT architectures differ in runtime behaviour, how fine-tuning and quantization affect deployment trade-offs, and how quality and resource use are measured.

3.1 Speech Translation as a Cascaded System

For local speech translation, transcription quality alone is not enough. The input is a time-varying speech signal, while the desired output is text in another language that must arrive quickly enough to support an ongoing interaction. We implement this task as a cascaded system, where an ASR model first produces a source-language transcript and an MT model then translates that transcript into the target language.

This structure differs from direct speech translation, where a single model maps speech directly to translated text. The cascaded design is useful because it allows specialised ASR and MT components to be selected, evaluated, and optimised independently. It also makes failures easier to diagnose, as transcription errors can be separated from translation errors, and total latency can be decomposed into stage-specific contributions (Sperber and Paulik, 2020).

Let x denote the input speech signal, z an intermediate transcription, and y the translated output. A cascaded speech translation system can be written conceptually as

$$P(y \mid x) = \sum_{z} P(y \mid z)\, P(z \mid x). \tag{3.1}$$

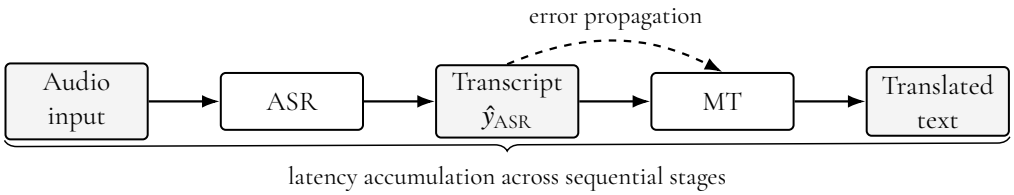

Figure 3.1: Conceptual structure of a cascaded ASR–MT speech translation system. The ASR stage produces the source transcript used by the MT stage, so recognition errors and latency both propagate through the pipeline.

This expression marginalizes over all possible transcriptions. In practice, retaining all hypotheses is usually infeasible because the computational cost grows with the number of ASR alternatives. The ASR model commonly produces a single hypothesis $\hat{z}$, which is passed to the MT model:

$$P(y \mid x) \approx P(y \mid \hat{z}), \qquad \hat{z} = \arg\max_{z} P(z \mid x). \tag{3.2}$$

This approximation makes inference practical, but it also exposes the main weakness of the cascade. Any error in $\hat{z}$ becomes input noise to the MT stage. Substitutions may alter meaning, deletions may remove important context, and insertions may introduce unwanted content that was not present in the speech. The MT model therefore receives input that can differ from the clean parallel text on which it was trained, which can degrade translation quality (Khayrallah and Koehn, 2018; Sperber and Paulik, 2020). This motivates the later comparison between clean reference-input translation and noisy ASR-input translation.

Latency accumulates in the same sequential way. Since MT depends on the ASR output, the total end-to-end processing time can be written as

$$T_{\text{e2e}} = T_{\text{ASR}} + T_{\text{MT}} + T_{\text{overhead}}, \tag{3.3}$$

where T_{ASR} is the recognition time, T_{MT} is the translation time, and T_{overhead} includes preprocessing, orchestration, and other system-level costs. A component that performs well in isolation may therefore still be unsuitable when integrated into the full pipeline.

3.2 Speaker Diarization

Practical speech translation systems often operate in conversations rather than isolated utterances. In such settings, the system must not only transcribe and translate speech, but also preserve information about speaker turns. Speaker diarization addresses this problem by assigning regions of an audio recording to unique speakers, commonly described as identifying "who spoke when?" (Park et al., 2021).

Diarization is relevant to speech-driven localisation because speaker attribution helps preserve conversational structure. Without it, translated text may retain the linguistic content but lose the turn-taking information that makes an interaction understandable. Diarization also introduces a deployment trade-off as it increases computation, may delay downstream ASR and MT, and can introduce segmentation errors. Incorrect speaker boundaries

can split utterances unnaturally or remove context that the ASR and MT components need. Such errors would then propagate through the same cascaded structure as recognition errors.

While we do not evaluate diarization experimentally, we included it here because it is relevant to practical multi-speaker deployment. The evaluated pipeline assumes utterance-level input without speaker attribution.

3.3 Automatic Speech Recognition

ASR is the upstream component of the evaluated speech translation pipeline. It converts speech into the source-language text that the MT model receives, so its output affects both transcription quality and downstream translation quality. Given an acoustic input x, ASR can be formulated as

$$\hat{y} = \arg\max_{y} P(y \mid x), \tag{3.4}$$

where y is a sequence of output tokens (Jurafsky and Martin, 2026). While this formulation is compact, the underlying task is difficult because speech is continuous, temporally structured, and highly variable across speakers, accents, speaking rates, and acoustic environments. The model must align a long sequence of acoustic frames with a much shorter sequence of discrete output tokens.

Modern ASR systems primarily address this challenge using end-to-end neural architectures, where a single architecture learns to map acoustic features to output tokens (Prabhavalkar et al., 2024). The two model families most relevant to this thesis are encoder-decoder models and transducer-based models. They primarily differ in how they generate output tokens, and those differences become noticeable when comparing transcription quality, latency, and deployment feasibility.

3.3.1 Encoder-Decoder ASR

Attention-based encoder-decoder ASR models generate output text autoregressively. The encoder maps acoustic features into a sequence of hidden representations, and the decoder generates output tokens one at a time while conditioning on the encoded speech and on previously generated tokens. The probability of an output sequence $y = (y_1, \ldots, y_T)$ conditioned on acoustic input x can factorised as

$$P(y \mid x) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, x). \tag{3.5}$$

This mechanism gives the decoder access to previous linguistic context, which can improve transcription fluency and help resolve ambiguity (Chan et al., 2016). However, the same property also affects deployment. Since tokens are generated sequentially, decoder-side computation cannot be fully parallelised across the output sequence. Larger encoder-decoder models may therefore improve WER while increasing latency, especially for longer utterances or unstable decoding cases.

Whisper

Whisper is a multilingual encoder-decoder Transformer ASR model trained on 680,000 hours of multilingual and multitask speech data (Radford et al., 2022). The model converts input audio into log-mel spectrogram features, processes them with a Transformer encoder, and generates text autoregressively with a Transformer decoder. This architecture makes Whisper a useful baseline family because the available model sizes represent different points on the accuracy-latency-memory spectrum.

A key feature of Whisper is that decoding can be conditioned on special control tokens, including task and language tokens (Radford et al., 2022). If the source language is supplied, the model can focus on transcription in the specified language, if it is not, language identification becomes part of the decoding process. This distinction matters for deployment because the system may not always receive a reliable external language label for each utterance.

Whisper's multilingual training makes it attractive for a six-language speech translation system, but its deployment behaviour must be measured. The smaller variants reduce memory and computation, but may produce higher WER and be more vulnerable to language-identification errors. The larger variants provide more model capacity, but their autoregressive decoder and larger parameter count can increase inference time and memory use. We therefore evaluate several Whisper variants rather than treating Whisper as a single model.

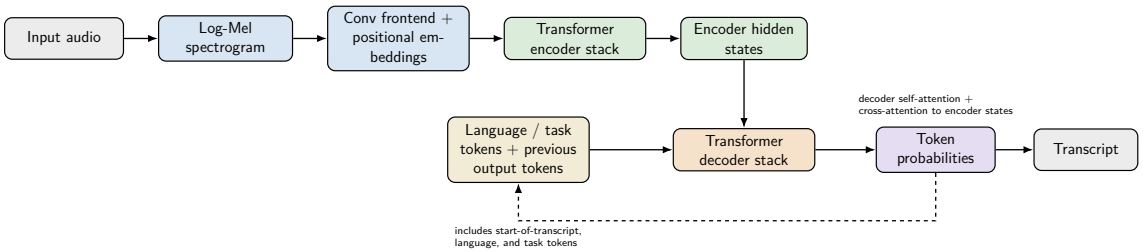

Figure 3.2: Simplified overview of the Whisper architecture as an encoder-decoder Transformer ASR model.

3.3.2 Transducer-Based ASR

Transducer-based ASR provides an alternative formulation to full-sequence encoder-decoder generation. A conventional recurrent neural network transducer consists of an encoder, a prediction network, and a joint network (Graves, 2012). The encoder processes the acoustic input and produces a sequence of speech representations. The prediction network conditions on output tokens that have already been emitted. The joint network then combines the acoustic representation from the encoder with the label context from the prediction network and estimates the distribution over the next decoding action.

The important difference from full-sequence encoder-decoder models is how they treat alignment. Transducer models learn how output tokens align with acoustic frames, including when to emit a token and when to advance through the input. This makes them naturally suited to incremental recognition, where output can be produced as speech is processed rather than only after a complete sequence has been encoded and decoded. This does not make transducer models automatically superior, but it makes them worth evaluating separately from encoder-decoder baselines.

FastConformer

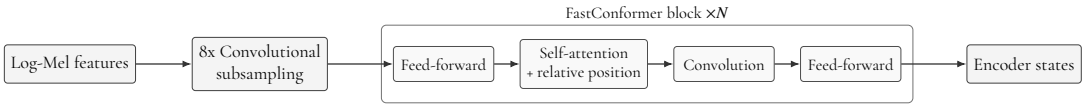

Figure 3.3: Simplified FastConformer encoder. Residual connections and layer normalisation are omitted for clarity.

FastConformer is an efficiency-oriented variant of the Conformer architecture for speech recognition. Like the original Conformer, it combines self-attention, which captures long range dependencies, with convolutional modules that model local acoustic structure (Gulati et al., 2020). FastConformer modifies this structure to reduce the computational cost of training and inference, most importantly by using a more aggressive downsampling scheme that shortens the acoustic sequence processed by later encoder layers (Rekesh et al., 2023).

Figure 3.3 should be read from left to right. The input audio is first converted into log-mel features. Convolutional subsampling then reduces the number of time steps before the deeper encoder stack. Each FastConformer block then applies feed-forward layers, self-attention, and convolution. The attention module helps the model use broader context across the utterance, while the convolution module preserves local acoustic patterns that are important for speech. The output is a shorter sequence of encoder states that can be consumed by the transducer decoder.

The cost of ASR inference is strongly affected by the length of the acoustic sequence processed by the encoder. For example, speech sampled at 16 kHz produces many frames even for short utterances, and later encoder layers can become expensive if the sequence remains long. Subsampling reduces the number of time steps passed through the deeper encoder stack, which can lower latency while preserving enough information for recognition.

The FastConformer design helps explain why a relatively large ASR model can still be competitive in real-time inference. Parameter count alone does not determine deployment feasibility. Effective sequence length, operator structure, runtime backend, and decoder behaviour also influence measured latency and memory use.

Token-and-Duration Transducer

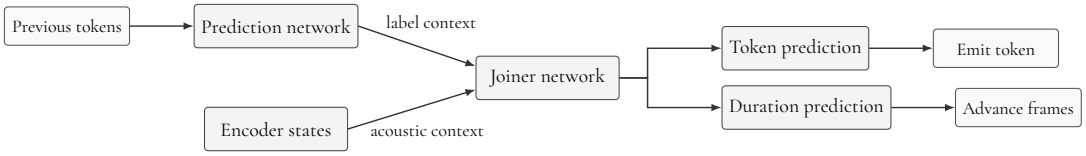

Figure 3.4: Simplified Token-and-Duration Transducer decoding. The token is emitted as output, and the duration determines how far decoding advances through the encoder sequence.

The Token-and-Duration Transducer (TDT) extends conventional transducer decoding by predicting both an output token and a duration (Xu et al., 2023). The duration represents how many frames are covered by the token and during inference this allows the decoder to

advance over multiple frames rather than processing the encoder output strictly frame by frame.

Figure 3.4 shows the role of the prediction and joiner networks in this process. The prediction network receives the previously emitted tokens and represents the current label history. The encoder states provide the acoustic context. The joiner network combines these two sources of information and produces both the next-token distribution and the duration prediction. The two outgoing paths in the figure therefore represent both what token to emit, and how far to advance over the sequence.

This mechanism can help deployment because it can reduce the number of decoding steps required for an utterance. If a model can emit a token and skip over the corresponding duration, it may reduce the amount of decoder work needed to process the speech signal. In a speech translation pipeline, a faster ASR stage leaves more latency budget for MT and system overhead.

Parakeet-TDT

Parakeet-TDT-0.6B-v3 combines a FastConformer encoder with a TDT decoder and has approximately 600 million parameters (Sekoyan et al., 2025). The model is trained on around 700,000 hours of ASR training data and supports 25 European languages. Its architecture is relevant as it is designed to provide strong recognition accuracy while reducing recognition latency through encoder subsampling and duration-aware transducer decoding.

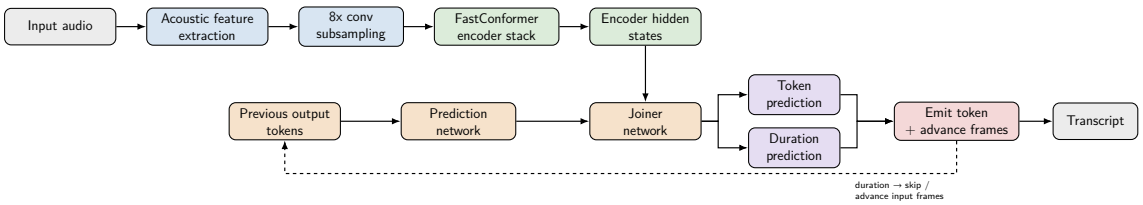

Figure 3.5: Simplified overview of the NVIDIA Parakeet architecture as a FastConformer-TDT ASR model.

Figure 3.5 summarises the complete Parakeet-TDT structure used in the ASR evaluation. The acoustic frontend converts input audio into features, the FastConformer encoder produces a shortened sequence of encoder states, and the TDT decoder generates the transcript by combining acoustic context with the previous output tokens. The prediction network models the already emitted token history, while the joiner network combines this label context with the encoder states and predicts both the next token and the duration used to advance through the input.

For this study, Parakeet has two roles. First, it is a candidate model in the initial model comparison, where it is compared against Whisper variants. Second, it becomes the focus of a deployment optimisation study, where we examine how backend selection, ONNX export, graph preprocessing, and dynamic INT8 quantization affect practical feasibility.

3.4 Machine Translation

The MT model converts the source-language transcript into target language text. In the evaluated cascade, MT is not a stand-alone text-generation task as the model receives ASR-generated transcripts. Its behaviour therefore determines both final translation quality and much of the end-to-end latency.

Neural machine translation (NMT) models estimate the probability of a target sequence y given a source sequence x:

$$\hat{y} = \arg\max_{y} P(y \mid x). \tag{3.6}$$

In cascaded speech translation systems, the translation model may receive text containing recognition errors, inconsistent casing, missing punctuation, or hallucinated fragments. Robustness to imperfect source text is therefore an important consideration for downstream translation quality (Khayrallah and Koehn, 2018).

3.4.1 Transformer-Based Translation

Modern NMT systems mostly adopt some sort of sequence-to-sequence architecture that learns to represent the source sentence and generate the target sentence token by token (Bahdanau et al., 2016). The encoder maps the source language input word tokens into a set of hidden representations, and the decoder produces the target sequence while conditioning on the source representation and previously generated target tokens.

Early sequence-to-sequence models compressed the entire source sentence into a fixed-length vector, which limited performance. Attention mechanisms address this shortcoming by allowing the decoder to dynamically focus on relevant encoder states during translation. This mechanism significantly improves translation quality by enabling dynamic alignment between source and target sequences, and has become a core component of modern NMT systems (Bahdanau et al., 2016).

The Transformer architecture extends this idea by relying entirely on attention mechanisms, rather than recurrence, to model dependencies between tokens (Vaswani et al., 2017). Self-attention enables the model to capture dependencies across the full input sequence and supports parallel computation during encoding. The Transformer computes attention as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{3.7}$$

where Q, K, and V denote the query, key, and value matrices, respectively, and d_k is the dimension of the keys used as a scaling factor to prevent the dot products from growing too large. As a result, Transformers have become the dominant architecture for modern NMT, including the models evaluated in this thesis. However, despite parallelism on the encoder side, decoding is sequential, which means that inference latency still depends on output length and decoding configuration.

3.4.2 Multilingual NMT and NLLB

Traditional NMT models typically require one model per language pair, making them difficult to scale across many languages. Multilingual NMT addresses this issue by training a single model to perform translation across multiple language pairs (Johnson et al., 2017). One common approach is to prepend a language token to the input, for example `<sv>`. This special token indicates to the model that its target language is Swedish. Multilingual NMT models have shown better performance, especially for low-resource language pairs where training data is sparse. This is due to the model having learned high level patterns from high resources language pairs. An example of a multilingual NMT model is NLLB, which supports over 200 languages, including many low-resource language pairs (Team et al., 2022).

3.4.3 Mixture Of Experts

MoE is a model architecture that replaces the standard FFN sublayer in a transformer with a collection of parallel feed-forward networks, referred to as experts, together with a learned routing network (Shazeer et al., 2017). MoE works by activating only a subset of the network for a given token. For each input token, the router computes a probability distribution over the experts and selects the top-K experts to process the token. The output is then computed as a weighted sum of the selected experts outputs. This allows the model to have significantly more parameters as only a subset of the parameters are used for a given token. Even with a large parameter count, the model maintains fast inference times due to the reduced number of active parameters per token.

3.4.4 Knowledge Distillation

Knowledge distillation is a technique used for model compression. It involves training a smaller student model to mimic a larger teacher model (Hinton et al., 2015). Rather than training a model with just the regular training data, the student model is trained with both the training data and the output probability distribution of the teacher model. The output probability distribution from the teacher model is referred to as soft targets and helps the student model learn uncertainty and relationships between outputs. Soft targets are produced by applying a softmax with a temperature parameter T to the teacher's logits z_i:

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{3.8}$$

The student is then trained using a weighted combination of two cross-entropy losses, one against these soft targets and one against the hard ground-truth labels. This allows the student model to gain a deeper understanding that only larger models can capture. This enables smaller models to achieve performance closer to larger models while maintaining lower inference time.

3.5 Fine-Tuning and Model Adaptation

Large language models typically contain hundreds of millions to hundreds of billions of parameters. Fine-tuning all parameters involves computing gradients for every weight, which requires considerable memory and compute resources. During full fine-tuning, the model is initialized with pre-trained weights and optimized by maximizing the following objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log(P_\Phi(y_t|x, y_{<t})) \tag{3.9}$$

When fine-tuning for a specific task like a language pair in an MT model, training all parameters often leads to catastrophic forgetting and an overall model performance decrease. Parameter-efficient fine-tuning (PEFT) addresses this issue by only updating a subset of the model weights. This reduces memory and compute resources while at the same time often increasing model accuracy.

3.5.1 Catastrophic Forgetting

Catastrophic forgetting or catastrophic interference refers to the unlearning or forgetting of knowledge that a model had prior to fine-tuning. This problem was first introduced by McCloskey and Cohen (1989), where they demonstrated that neural networks struggle to learn new information without losing previously learned knowledge. This is especially common when a model is fine-tuned on a very specific task, like a unique language pair for a MT model. Fine-tuning on a single language pair can degrade performance on other languages, as parameter updates shift the model toward the new target distribution. PEFT addresses this issue by freezing the original model weights during fine-tuning, which reduces interference with previously learned knowledge while still allowing adaptation to a specific task.

3.5.2 Parameter Efficient Fine-Tuning

PEFT refers to a method of fine-tuning where only a subset of the weights are updated rather than all weights. Which reduces memory and computational requirements and enables smaller, more task-specific fine-tuning setups.

A related approach to PEFT is partial fine-tuning, where a subset of the original model weights are updated. This is typically done by unfreezing certain layers in the network, often starting with the final layers while keeping earlier layers frozen. Although simpler than full fine-tuning, this approach still updates a large number of parameters compared to PEFT methods.

There exist several PEFT methods. Adapter layers introduced in Houlsby et al. (2019) proposes to add a small number of trainable parameters between existing layers, while freezing all the original weights. The authors showed that they were able to get near full fine-tuning performance while only training on a fraction of the weights. One disadvantage of adapter layers is the increase in the number of model parameters, which can lead to slower inference times. Another PEFT method is prefix-tuning, introduced by Li and Liang (2021).

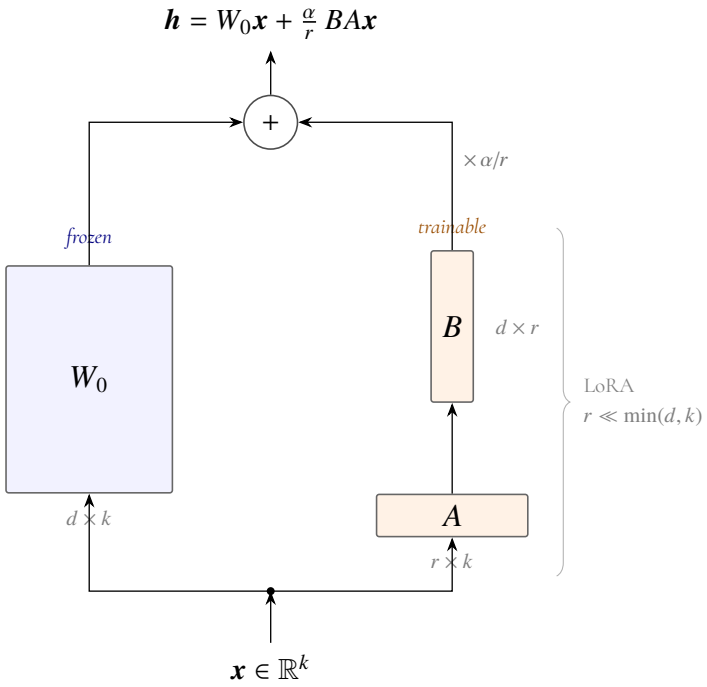

Figure 3.6: LoRA applied to a single linear layer.

Prefix-tuning freezes the pre-trained model and learns a small set of learned continuous prefix vectors that are injected into the transformer attention mechanism. This allows the model to adapt to specific tasks while updating only a very small number of parameters.

3.5.3 Low-Rank Adaptation

Low-Rank Adaptation (LoRA) is a PEFT method introduced by Hu et al. (2021) that freezes the pre-trained model weights and introduces trainable low-rank decomposition matrices in selected model layers. Instead of updating all the model weights, LoRA creates two rectangular matrices that are multiplied together during inference and added with the original weight matrix. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA represents it as a low-rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training W_0 is frozen and only A and B are updated. The A matrix is initialized with a random Gaussian, while B is initialized with zeros. LoRA can be applied to different matrices, in the paper Hu et al. (2021) they apply LoRA to the attention matrices W_Q, W_K, W_V, W_O. The best results are obtained when applying LoRA to all attention matrices, although it can also be restricted to specific matrices to further reduce the number of trainable parameters.

To illustrate the benefit of LoRA, suppose W_0 has dimensions $10,000 \times 10,000$, which corresponds to 100 million parameters. If the rank r is set to 8, that would mean that A has $8 \times 10,000$ parameters and matrix B contains $10,000 \times 8$ parameters. This results in only 160,000 trainable parameters which is far less than the original 100 million parameters.

One of the key benefits of LoRA is that it does not increase inference cost after the learned weights are merged. Once training is completed, the matrices A and B are merged with the frozen weights as $W = W_0 + BA$. This means that W_0 and the resulting W have the same dimensions, so the original model architecture remains unchanged, resulting in no

additional inference latency. Another advantage of LoRA is that because we simply add our new weights with the original it is very easy to swap out the fine-tunes during inference. This enables loading task-specific fine-tuned adapters during inference, allowing the model to be adapted to different use cases.

3.6 Tokenization and Decoding

Both ASR and MT systems typically operate on subword units rather than full words. Full-word vocabularies are difficult to scale across languages, while character-level representations produce excessively long sequences. Subword methods such as Byte Pair Encoding (BPE) and Unigrams provide a balance by decomposing words into smaller units that can be recombined during decoding (Sennrich et al., 2016; Kudo and Richardson, 2018).

As well as managing vocabulary, tokenization also affects system behaviour. It determines sequence length, influences decoding complexity, and shapes how outputs are reconstructed and evaluated. Even though evaluation metrics operate on full words or sentences, the underlying tokenization remains important for both model efficiency and error behaviour.

At inference time, exact search over all possible output sequences is infeasible, so practical systems use approximate decoding strategies. Greedy decoding selects the most likely token at each step (Jurafsky and Martin, 2026):

$$\hat{y}_t = \arg\max_{v \in V} P(v \mid \hat{y}_{<t}, x), \tag{3.10}$$

where V is the output vocabulary. Greedy decoding is computationally cheap, but it may make locally optimal choices which can be detrimental to the final sequence quality.

Another strategy is beam search, which keeps B partial hypotheses at each decoding step instead of only one. This can improve output quality by reducing the effect of early local decisions, but it increases computational cost. For output length T, vocabulary size $|V|$, and beam width B, the dominant search cost scales approximately as

$$O(B \cdot T \cdot |V|), \tag{3.11}$$

before implementation-specific optimisations.

Decoding configuration is therefore part of the overall deployment trade-off. Larger beams may improve translation quality in some settings, but they also increase latency and CPU pressure. For an interactive local pipeline, decoding must be selected not only for quality, but also for its effect on end-to-end response time.

3.7 Quantization for Neural Inference

Quantization is a numerical approximation technique used to reduce the precision of neural network parameters and, in some cases, intermediate activations. Its main purpose is to reduce memory consumption and to improve inference efficiency by enabling lower-precision arithmetic. Rather than representing values in 32-bit floating point (FP32) format, a quantized model maps them to a discrete integer range, typically an 8-bit integer (INT8) format. In effect, quantization maps a wide range of values into a more compact one, which sounds trivial but requires a clearly defined mapping process.

The quantization experiments in this thesis use post-training quantization (PTQ), where the pretrained model is transformed after training rather than retrained with quantization-aware objectives, allowing us to apply it to existing exported models. However, quantization cannot guarantee an improvement of all models or operators. Lower precision can reduce memory traffic and arithmetic cost, but it can also introduce numerical error, conversion overhead, or inefficient execution paths.

3.7.1 Affine Quantization

A common PTQ scheme uses an affine mapping between real-valued tensors and integer tensors. Given a real-valued tensor x, quantization maps each value to an integer range $[q_{\min}, q_{\max}]$ using a scale $s > 0$ and a zero-point z:

$$q = \text{clamp}\left(\left\lfloor \frac{x}{s} \right\rceil + z, q_{\min}, q_{\max}\right). \tag{3.12}$$

The corresponding dequantized approximation is

$$\hat{x} = s(q - z). \tag{3.13}$$

The scale s determines the step size between adjacent quantized integer values, while the zero-point z allows the real value zero to be represented exactly in the quantized domain (Jacob et al., 2017). This mapping introduces approximation error through rounding and, when values exceed the representable range, through clipping.

For values that are not clipped, the absolute rounding error is bounded by approximately half a quantization step:

$$|\epsilon_q| \leq \frac{s}{2}. \tag{3.14}$$

This shows the central trade-off of quantizing. A smaller scale gives finer numerical resolution but covers a narrower real-valued range. A larger scale covers a wider range, but represents small differences less accurately.

3.7.2 Static and Dynamic Quantization

Two common forms of PTQ are static and dynamic quantization and they primarily differ in how activation quantization parameters are obtained. Static quantization computes the quantization parameters for model weights and activation before inference. Weights are quantized directly from the trained model parameters, while activation ranges are estimated using a calibration dataset that represents the expected deployment inputs. Once calibration is complete, the runtime can execute more of the graph using integer arithmetic because the activation scales and zero points are known in advance (Gholami et al., 2021). This can improve latency when the runtime provides efficient kernels, however, static quantization depends on calibration quality. If the calibration set does not reflect deployment inputs, activation ranges may be poorly estimate, leading to excessive clipping.

Dynamic quantization differs from static quantization in the sense that it defers parts of this process to inference time. The weights are quantized in advance, but the activation ranges are computed dynamically for each input. This makes dynamic quantization easier to

apply because it does not require a calibration dataset. It is often useful for operators such as matrix multiplication, where weights are fixed and the computational savings from lower-precision arithmetic can outweigh the cost of dynamically quantizing activations (ONNX Runtime, 2025).

The drawback is that dynamic quantization introduces runtime work. The system may need to compute activation ranges, quantize activations, execute the integer operator, and dequantize outputs or pass them through quantize dequantize boundaries. If the operator is not a dominant runtime bottleneck, if the tensor shapes are unfavourable, or if the backend lacks efficient kernels for the quantized form, this additional work can outweigh the arithmetic savings.

3.7.3 Operator-Level Quantization Effects

The effect of quantizing a model not only depends on precision, but also on which operators are quantized. In ONNX, `MatMul` represents standard matrix multiplication, similar to `numpy.matmul`, while `Gemm` represents a more general affine matrix operation, $Y = \alpha A'B' + \beta C$, where A and B may be transposed and C is an optional additive term (ONNX, 2026b,a). In exported neural networks, these operators commonly represent the dense linear algebra used in attention projections, feed-forward layers, and output projections.

Dense operators such as `MatMul` and `Gemm` are often good quantization targets because they can dominate much of the arithmetic work in Transformer-style models and are commonly supported by optimised INT8 CPU kernels (ONNX Runtime, 2025). Quantizing these operators can reduce memory traffic and improve arithmetic throughput when the runtime provides efficient kernels.

Convolutional operators can behave differently. Although convolutions can benefit from quantization on suitable hardware and runtimes, dynamic quantization can also introduce additional overhead from activation scaling, zero-point handling, and quantization/dequantization conversions. If the convolutional layers do not dominate the runtime bottleneck, or if the available INT8 convolution kernels are less efficient for the relevant shapes, quantizing them may fail to improve latency. It may also introduce additional approximation error in components that are more sensitive to reduced precision.

The benefit of quantization is therefore not only guaranteed by lower precision alone, but also by which operators are quantized. A model may become smaller without gaining speed, and a broader quantization scope may degrade accuracy or latency if it introduces inefficient conversion paths or excessive numerical error. The focused ASR deployment study in this thesis evaluates quantization not only as a choice between FP32 and INT8, but also as a question of inference backend, export pipeline, and operator scope. This distinction is necessary because the most efficient deployment configuration may be a selectively quantized model rather than the most aggressively quantized one.

3.8 Performance Metrics

This thesis evaluates feasibility using quality, latency, CPU utilisation, and memory consumption. We report them together because no single metric is sufficient. A model with low

Word error rate (WER) may still be unsuitable if it exceeds the memory budget, while a fast model may be unsuitable if its transcription errors degrade downstream translation.

3.8.1 Real-Time Factor and Latency

Real-time factor (RTF) measures how quickly a system processes audio relative to the duration of the input audio:

$$\text{RTF} = \frac{t_{\text{process}}}{t_{\text{audio}}}, \tag{3.15}$$

where t_{process} is the measured processing time and t_{audio} is the duration of the audio signal. An RTF below 1 means that processing is faster than the duration of the input.

For a speech translation pipeline, RTF determines whether the system can support an ongoing interaction or only serve as an offline transcript. Since latency distributions can be long-tailed, the thesis reports both median and 95th percentile RTF. The median describes typical behaviour, while the 95th percentile captures tail or worst case latency that may affect user-perceived responsiveness.

For the MT stage, runtime is also normalized by the duration of the corresponding source audio. Even though MT processes text, this allows us to compare ASR, MT, and full-pipeline latency within the same environment. Throughput measures such as TPS can still be useful for comparing text-generation configurations, but they do not directly answer whether translation keeps pace with the source utterance.

3.8.2 Tokens Per Second

Tokens per second (TPS) measures the output speed from a large language model (LLM). To compare different MT models in the context of inference speed we use TPS, which is described by the following function:

$$\text{TPS} = \frac{N_{tokens}}{t_{MT}}, \tag{3.16}$$

where N_{tokens} denotes the number of tokens and t_{MT} denotes the time it takes. This is a good metric for comparing MT models, but is hard to compare with ASR metrics.

3.8.3 Word Error Rate

To evaluate speech recognition systems you compare the models output to target text, and annotate any errors in the transcription. There are three error categories, the first being substitutions, this means that the model has substituted a word for an incorrect word, for example "throw" to "threw". The second is insertion which means that the model has inserted a word into the transcription. The third is deletion which means that the model has removed a word in the transcription. The most common evaluation metric is the WER, which measures the ASR errors relative to reference transcriptions:

$$\text{WER} = \frac{\text{S} + \text{D} + \text{I}}{\text{N}}, \tag{3.17}$$

where S, D, and I are the number of substitutions, deletions, and insertions, and N is the number of reference words (Jurafsky and Martin, 2026).

WER is useful because it gives a measure of transcription error, but in practice it depends strongly on text normalization. Differences in casing, punctuation and tokenization can significantly affect the computed error rate without reflecting a meaningful recognition difference. This is particularly important in multilingual settings, where writing conventions across languages and model output formatting differ. To ensure fair and consistent evaluation, both reference and predicted transcriptions are consistently normalized across before WER computation.

3.8.4 BLEU

Evaluating MT systems requires objective, reproducible metrics that quantify the correlation between a system-generated translation and human judgement of quality. BLEU, introduced by Papineni et al. (2002), evaluates translation quality using modified n-gram precision combined with a brevity penalty to discourage overly short outputs:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{3.18}$$

where p_n is the modified precision for n-grams of order n, w_n is the corresponding weight, and BP is the brevity penalty.

3.8.5 CPU and Memory Metrics

In resource-constrained deployments, computational efficiency is as important as system accuracy. CPU utilisation and memory consumption indicate whether a model can be deployed reliably under local hardware constraints. CPU utilisation is reported relative to one logical CPU core, 100% corresponds to one fully utilized logical CPU core, so a four-core limit corresponds to a sustained capacity of 400% (Docker Inc., 2026b).

Memory consumption is measured using Resident set size (RSS), which represents the physical memory occupied by the process. Peak memory is particularly important because a model can fail under a memory limit even if its average memory use appears acceptable.

Both CPU utilisation and memory consumption directly influence achievable RTF and system scalability. High CPU utilisation may reduce latency but can compromise stability, whereas excessive memory consumption limits deployability on resource-constrained systems with limited RAM.

3.8.6 Floating-Point Operations

Floating-point operations (FLOPs) provide a hardware-independent estimate of computational complexity and can be used to estimate the theoretical workload of running neural network inference. For a model composed of L layers, the total operation count can be expressed as:

$$\text{FLOPs}_{\text{model}} = \sum_{\ell=1}^{L} \text{FLOPs}_{\ell}. \tag{3.19}$$

Although FLOPs are useful for comparing theoretical workload, they do not by themselves determine real-world latency. Observed runtime also depends on factors such as memory access patterns, operator implementation, hardware acceleration, and inference-engine optimizations.

3.9 System-Level Feasibility

The central evaluation problem in this thesis is whether a cascaded ASR-MT pipeline can operate locally under deployment-oriented constraints. To do this several objectives have to be compared at once. Transcription quality affects the input seen by the MT model. while translation quality determines whether the final output preserves the intended meaning. Latency determines whether interaction can occur close to real time. CPU and memory consumption determine whether the system can run with the available local resources.

These objectives can conflict. A larger model reduce errors but increase memory usage or latency and vice-versa. Quantization may reduce memory consumption and improve speed, but overly broad quantization can degrade ASR quality. Similarly, MT fine-tuning may improve weak language direction while reducing performance on others if the adaptation is not balanced.

In short, improvements in one dimension often come at the expense of another, a configuration is not suitable simply because it performs well on one metric. It must be evaluated in terms of the combined behaviour of accuracy, latency, CPU usage, memory usage, and integration effects. The conclusions are consequently limited to the evaluated models, languages, datasets, runtime stack, and resource constraints. They indicate how specific design choices behave under controlled local inference conditions, rather than establishing general superiority of one model family or optimization method.

Chapter 4

Datasets

The evaluation requires data that links speech, source-language text, and target-language references in order to compare ASR, MT, and the complete cascaded pipeline. We therefore use a FLEURS-derived corpus as the main evaluation dataset and a TED2020-derived corpus for MT fine-tuning. FLEURS provides aligned multilingual speech and text for controlled component-level and end-to-end evaluation, while TED2020 provides larger-scale parallel text for adapting the MT model to selected language directions.

4.1 Dataset Requirements

The evaluation corpus had to satisfy three requirements. First, it needed to include speech recordings, reference transcriptions, and multilingual text references for the same underlying sentence content. This made it possible to evaluate the ASR component, the MT component, and the complete ASR-MT cascade using consistent inputs. Second, it had to cover English, Swedish, Dutch, French, German, and Spanish. Third, it had to contain utterances short enough to reflect interactive speech-driven use cases rather than long-form offline transcription.

A further requirement was consistency across experiments. We compare model architectures, runtime backends, quantization settings, fine-tuning strategies, and pipeline conditions. If the evaluation data changed between stages, differences in measured performance would be harder to interpret. We therefore used a shared FLEURS-derived evaluation framework across the main ASR, MT, and end-to-end experiments. Speech-based evaluations used a globally filtered valid-audio subset, while stand-alone MT evaluation used the corresponding text corpus without audio-based exclusions.

Potential overlap between evaluation data and the data used to pretrain ASR and MT models. The pretraining corpora for modern multilingual models are large, aggregated from multiple sources, and not always documented at the level needed to rule out overlap completely. The results should therefore be interpreted as comparisons between models and de-

ployment strategies under identical evaluation conditions, not as a strict estimate of generalisation performance.

4.2 Google FLEURS

The main evaluation corpus was derived from Google FLEURS, released under the CC-BY-4.0 licence (Conneau et al., 2023; Creative Commons, 2021a). We chose FLEURS as it covers 102 languages with approximately 12 hours of audio and corresponding transcriptions per language. Of these 102 languages, 6 were used to create the initial corpus seen in Table 4.1, these being: English, Swedish, Dutch, German, Spanish, and French. These languages were selected based on availability and our understanding of the languages.

A key advantage of the FLEURS dataset is that it consists of n-way parallel data, which means that each sentence has a corresponding sentence in all the other languages while sharing the same unique id, a structure that allowed us to use the dataset for evaluating both the MT and ASR components (Conneau et al., 2023). Using this shared identifier, we filtered the data so that the evaluation corpus contained 1214 parallel utterances per language.

The audio recordings were used as input to the ASR component, while the corresponding human-generated transcriptions served as reference text for evaluating transcription accuracy. The aligned translations across the selected languages were additionally used as reference targets for evaluating the MT component of the system as well as the clean-reference branch of the integrated pipeline evaluation. This enabled consistent evaluation of both components and the pipeline within the same experimental framework.

Language	Utterances	Total Dur. [h]	Mean Dur. [s]	P95 Dur. [s]	Total Words
Dutch	1214	3.197	9.481	15.54	26392
English	1214	3.407	10.103	16.36	25020
French	1214	3.861	11.449	17.28	28999
German	1214	3.641	10.798	16.62	24943
Spanish	1214	3.762	11.156	17.52	24676
Swedish	1214	4.237	12.563	20.94	23395
Total	7284	22.106	10.925	17.7	153425

Table 4.1: Statistics for the full FLEURS-derived text corpus before audio-based filtering. Speech-based experiments use the filtered valid-audio subset described below.

4.2.1 Filtering and Evaluation Subsets

During dataset inspection, we found that a subset of the FLEURS utterances contained almost no speech content. We filtered these samples using voice activity analysis and excluded

them from speech-based evaluation. Since FLEURS uses shared sample identifiers across languages, we excluded the affected sample IDs globally from the speech evaluation subset. This preserved the parallel structure of the data across the six languages but reduced the number of samples per language to 1017.

In the experiments we therefore used the corpus in two forms. The stand-alone MT evaluation used the full text corpus, because those experiments evaluated translation independently of the audio signal. In contrast, the ASR and integrated pipeline evaluations used the globally filtered valid-audio subset. Within the pipeline evaluation, both the clean reference-input MT condition and the noisy ASR-input MT condition used this same filtered subset. The clean-noisy comparison is therefore paired sample-wise, while the standalone MT benchmark is a broader component-level text translation benchmark.

4.3 TED2020

To fine-tune our MT model, we construct a dataset using TED2020, released under the CC-BY-NC-ND-4.0 licence (Reimers and Gurevych, 2020; Creative Commons, 2021c). The TED2020 dataset consists of approximately 4,000 TED talks and is available in over 100 languages. Most importantly the six languages chosen from the FLEURS dataset were also present in the TED2020 dataset. This meant that regardless of which language pairs performed poorly, our dataset would cover it.

Opposed to the read speech of the FLEURS dataset, TED2020 contains speech that is closer to natural conversational use. The topics of the TED talks vary from complex medical presentations to more casual talks. When creating the fine-tuning dataset, we first clean the TED2020 dataset. We remove all examples containing fewer than three words. We also check the length ratio between source and target texts. If one text is more than three times longer than the other, we remove the sample, as a valid translation is unlikely to have such a severe length mismatch. We also filter out empty translations. We then cap the number of samples per language pair at 50,000. Leaving us with 30 language pairs with 50,000 sentences per pair, resulting in a dataset containing 1.5 million sentences. The last thing we do is split the data into a training and evaluation dataset, the evaluation dataset will be 5% of the total dataset. Resulting in a training dataset containing 1.425 million sentences and an evaluation dataset containing 75,000 sentences.

4.4 Dataset Limitations

The selected datasets support controlled evaluation, but they do not represent all conditions relevant to deployment. FLEURS consists of read speech and does not fully capture spontaneous conversation, overlapping speech, interruptions, background noise, speaker turn-taking, or domain-specific vocabulary. Since we evaluate segmented utterances rather than complete conversations, the evaluation also does not test diarization or multi-speaker interaction handling.

TED2020 is also limited by domain as it is derived from prepared talks and subtitle text, not the target deployment setting. Fine-tuning on TED2020 may therefore improve some translation directions without guaranteeing improvements for on-device speech interactions.

The six-language subset also limits generalisation. English, Swedish, Dutch, French, German, and Spanish represent a controlled multilingual evaluation setting, but they do not represent all languages, language families, accents, or deployment domains. The conclusions should be read as evidence for the evaluated language set and experimental setup, not as a general claim about multilingual speech translation.

4.5 Ethical and Licensing Considerations

The study uses existing public datasets rather than newly collected recordings from participants. Only data required for the experiments was retained in the processed corpora. Any speaker-related metadata was not used as an experimental variable, and we make no attempts to infer speaker identity.

FLEURS is used under the CC-BY-4.0 licence, while TED2020 is used under the CC-BY-NC-ND-4.0 licence (Creative Commons, 2021a,c). These licences are compatible with research use in this study, but the TED2020 includes non-commercial and no-derivatives restrictions that should be considered before use outside the context of this thesis.

Chapter 5
Methodology

5.1 Experimental Design

Local speech translation was evaluated as a system-level problem rather than a sequence of isolated model benchmarks. Each stage was designed to identify whether failure comes from recognition quality, translation quality, latency, memory use, or integration effects. We first evaluated ASR and MT separately, then optimised the strongest ASR candidate for local inference, fine-tuned the strongest MT candidate for weak language directions, and finally evaluated the integrated ASR-MT pipeline under clean and noisy input conditions.

We designed the experiment like this to separate failure modes. ASR can fail through recognition errors, MT can fail through poor translation or slow decoding, and the complete pipeline can fail because latency and memory accumulate across components. A configuration was therefore not only selected because it performs well on one metric. Instead, the evaluation considers transcription quality, translation quality, latency, CPU utilisation, memory consumption, and integration effects together.

The MT component was evaluated in two contexts. The stand-alone benchmark used the full FLEURS-derived text corpus to measure component-level translation quality. The integrated pipeline evaluation used the globally filtered valid-audio subset for both clean reference-input MT and noisy ASR-input MT. This design keeps the clean-noisy pipeline comparison paired, while allowing the stand-alone benchmark to use all available text examples.

5.2 Runtime Environment

Deployment-oriented experiments were executed in Docker containers running inside a Colima virtual machine. The main constrained setting used four CPU cores and an 8 GiB container memory limit. We also introduced a stricter 4 GiB memory setting in the ASR feasibil-

ity study to test whether the evaluated models remained operational under a smaller memory budget.

The container limits were configured using Docker CPU and memory controls and monitored using `docker stats` (Docker Inc., 2026a,b). Thread-level parallelism was controlled to reduce variation caused by runtime libraries. The number of intra-operation threads was aligned with the CPU limit when possible, and inter-operation parallelism was restricted to avoid uncontrolled parallel execution between operators. For ONNX Runtime experiments, graph optimisation was enabled and the session configuration was kept consistent across runs (ONNX Runtime, 2025).

The experiments should be interpreted as deployment-oriented container benchmarks, not as measurements on final embedded hardware. The controlled environment supports comparisons between model configurations, but it does not capture all properties of a physical device, such as thermal throttling, power consumption, accelerator availability, or real-time operating-system scheduling.

5.3 Preprocessing and Normalization

All FLEURS audio was loaded as mono waveform data and processed at 16 kHz. Each utterance was evaluated independently, and no batching or concatenation of utterances was used during latency measurement. This was done to represent the per-utterance response time in the expected deployment scenario, where speech is received, transcribed, translated, and returned as a single interaction.

For ASR evaluation, we normalized both reference transcripts and model hypotheses before computing WER. The normalization procedure applied Unicode NFKC normalization, lowercasing, punctuation and symbol removal, and whitespace normalization. We applied the same procedure to all ASR models and languages in order to reduce the influence of model formatting differences that were not central to recognition.

For MT and integrated pipeline evaluation, the text preprocessing was lighter. References and hypotheses were lowercased before scoring, while tokenization-sensitive metric handling was delegated to SacreBLEU. ASR-specific punctuation and symbol removal was not applied to MT outputs because punctuation and surface form can carry translation-relevant information.

5.4 Metric Implementation

WER was computed using the JiWER library (Jitsi, 2025), and we report both per-language and corpus-level WER. Per-language WER shows whether a model behaves unevenly across the selected languages, while corpus-level WER summarizes overall transcription quality across the evaluation set.

MT quality was measured using SacreBLEU with the international tokenizer to compute BLEU (Post, 2018; Papineni et al., 2002). Scores were computed at the corpus level for each directed language pair. To obtain the average BLEU score per language, we calculated the mean across its directed pairs. Because every language pair contained an equal number of samples, the macro-average and mean are identical. Self-translations were treated separately

from cross-lingual translation as they do not represent the same task as translation between different languages.

For the integrated pipeline, MT quality was evaluated under two input conditions. In the clean condition, we fed the MT model clean reference source-language transcripts from the evaluation dataset. In the noisy condition, the MT model received ASR-generated transcripts. The difference between these two estimates the loss in translation quality caused by recognition errors entering the MT stage.

5.5 Timing and Resource Measurement

Each resource benchmark included 5 warm-up samples before measurements. These samples were processed with the same inference procedure as measured samples but were excluded from the final metrics. We included warm-up to reduce the effect of one-time initialisation costs, such as graph preparation, memory allocation, tokenizer loading, and kernel initialisation.

Timing was measured using Python's high-resolution performance counter. For component-level evaluation, timing covered one inference pass and was logged per utterance. For the integrated pipeline, end-to-end latency was measured from the start of ASR processing until completion of the translated output. Thereafter RTF was computed as

$$\text{RTF} = \frac{t_{\text{process}}}{t_{audio}}, \tag{5.1}$$

where t_{process} is the measured processing time and t_{audio} is the duration of the source utterance. For the ASR, t_{process} corresponds to ASR processing time. When it came to computing the same metric for the MT component and the full pipeline, the denominator remained the duration of the source utterance, even though MT itself operates on text. This allowed us to compare the ASR, MT, and the pipeline latency within the speech translation setting.

Memory was reported primarily as RSS. Peak memory was especially important since a model that exceeds the memory budget may fail even though its average memory use is lower. If a configuration failed to initialize or complete inference within the imposed memory limit, it was marked as infeasible rather than excluded from the reporting.

CPU utilisation was sampled during the benchmark window. Since Docker reports CPU use relative to one logical CPU core, a four-core limit corresponds to 400% sustained utilisation. We report mean and 95th percentile CPU utilisation to describe typical and high-load behaviour during inference.

5.6 ASR Evaluation

The ASR evaluation was designed to measure both transcription quality and deployment feasibility. Since ASR provides the input to the MT stage, poor recognition can degrade the final translation even if the MT model performs well on clean text. At the same time, an accurate ASR model is not useful for local speech translation if it exceeds the memory budget or fails to process speech faster than real time.

5.6.1 ASR Model Selection

The ASR models were selected according to multilingual language support, reported recognition performance, availability, and expected deployment feasibility. Each model had to support English, Swedish, Dutch, French, German, and Spanish. Models exceeding approximately one billion parameters were excluded because we expected their memory footprint and latency to be unsuitable for the constrained real-time deployment setting.

Within these limits, the models listed in Table 5.1 were selected for the initial comparison, representing different points on the accuracy-efficiency spectrum. The Whisper variants, provide a family of multilingual encoder-decoder models with increasing model capacity (Radford et al., 2022). Parakeet-TDT-0.6B-v3 represents a Transducer-based architecture intended for multilingual ASR (Sekoyan et al., 2025). This selection made it possible to compare both scaling behaviour within a model family as well as architectural differences between model families

Model	Parameters [M]	Disk Size [MB]	Precision	GFLOPs
Whisper-tiny	37.8	143	FP32	51.4
Whisper-base	72.6	277	FP32	135.5
Whisper-small	241.7	925	FP32	604.6
Whisper-medium	763.9	2960	FP32	2143.6
Parakeet-TDT-0.6B-v3	627.0	2394	FP32	146.1

Table 5.1: Architectural and computational characteristics of the evaluated ASR models.

We used the model specifications to understand the measured accuracy and runtime results rather than to determine deployment suitability directly. Parameter count, disk size, and theoretical operation count alone do not determine inference latency or memory use, since runtime behaviour also depends on architecture, operator implementation, decoding procedure, framework overhead, and hardware execution characteristics.

5.6.2 Initial ASR Benchmark

The initial ASR benchmark evaluated the selected pretrained models on the FLEURS-derived evaluation corpus before imposing the full deployment-oriented benchmark setting. This stage was used to estimate transcription quality and identify which models should be retained for constrained evaluation.

For the Whisper models, the source language was provided during this initial benchmark. This configuration was set so that the benchmark primarily measured transcription quality rather than language identification. Apart from that, no task-specific fine-tuning or decoding optimisation was applied, so the results reflect out-of-the-box model behaviour. The resulting transcripts were normalized using the previously described procedure, aligned with the reference transcripts, and scored using WER.

Based on the results, Whisper-base was later excluded because it occupied an intermediate position between Whisper-tiny and Whisper-small and was unlikely to provide additional insight into the deployment trade-off. All of the other models were retained because they represented more distinct operating points across accuracy, latency, and memory use.

5.6.3 Constrained ASR Benchmark

While the baseline evaluation provided an upper bound on transcription accuracy, it did not reflect the conditions under which the system was intended to operate. We therefore introduced resource constraints to assess whether the observed performance could be sustained in a realistic deployment setting. This experiment evaluated Whisper-tiny, Whisper-small, Whisper-medium and Parakeet under four CPU cores and an 8 GiB memory limit. Then we repeated the experiment under a 4 GiB limit to test even stricter feasibility.

The constrained evaluation deviated from the initial benchmark in one important respect, the Whisper variants were no longer provided the spoken language. We made this choice because the intended deployment scenario cannot assume that the source language is externally specified for every utterance. As a result, the constrained Whisper results include both transcription errors as well as errors introduced by automatic language identification. This makes the constrained setting more representative of deployment, but less directly comparable to the initial accuracy-only benchmark.

For each model and configuration, we recorded transcription output, latency, RTF, CPU utilisation, and process memory. If a model failed under the memory limit, it was marked as infeasible.

5.6.4 Focused Parakeet Optimisation Study

The constrained evaluation indicated that Parakeet offered the strongest overall balance between transcription accuracy and inference speed under the 8 GiB memory constraint, but its memory footprint left limited deployment headroom. To better understand how these limitations could be mitigated in practice, we conducted a focused deployment optimisation study to evaluate how backend selection, exported graph preparation, numerical precision, and quantization scope affected Parakeet's accuracy, latency, and memory footprint.

Inference Backends

NVIDIA NeMo was used as the reference inference framework for Parakeet because it provides the native implementation of the model and its speech recognition pipeline (Kuchaiev et al., 2019; NVIDIA, 2026). In this configuration, Parakeet was executed through the original NeMo-based stack, including model loading, acoustic preprocessing, and decoding. This configuration therefore serves as the baseline against which deployment-oriented alternatives were compared.

To evaluate deployment-oriented inference, Parakeet was also executed using ONNX runtime. ONNX provides a framework-independent representation of the exported computation graph, while ONNX Runtime is the inference used to execute that graph (ONNX, 2026c; Microsoft, 2026). This distinction is important because ONNX itself is not an execu-

tion backend. The runtime determines how the graph is optimised, which kernels are used, and which PTQ workflows are available.

The focused Parakeet study varied four factors:

- **Backend:** native NeMo or ONNX Runtime inference.
- **Preparation pipeline:** original NeMo execution, vendor-provided pre-exported ONNX, locally exported ONNX, or ONNX graph preprocessing before quantization.
- **Precision:** FP32 or INT8.
- **Quantization scope:** no quantization, broader supported-operator quantization, MatMul/Gemm quantization, MatMul/Gemm/Conv quantization, or encoder-only variants of these scopes.

We introduced these factors because they correspond to practical optimisation choices for on-device inference. Switching from native NeMo inference to ONNX Runtime could reduce framework overhead, expose graph-level optimisation and improve runtime efficiency. Quantization can further reduce memory memory consumption and computational cost, but affects numerical precision and may therefore affect recognition accuracy. The preparation pipeline was treated as a separate factor because the exported graph structure can influence which operators can be quantized and how aggressively graph-level optimisations are applied.

Dynamic INT8 Quantization was applied using the ONNX Runtime post-training quantization workflow (ONNX Runtime, 2025). For the selectively quantized variants, we restricted quantization to the operator groups listed in Table 5.2. The encoder-only variants were only quantized in the exported encoder graph, while full-graph variants were quantized across the exported model graphs. All variants were evaluated under the same four-core CPU and 8 GiB memory limit, using the same dataset, normalization, warm-up procedure, timing method, and resource monitor.

5.7 Machine Translation

The MT component was evaluated based on its ability to provide high quality translations, as well as its computational footprint. Based on the system requirements, the MT model had to support many-to-many translation. Model selection and subsequent fine-tuning focused on improving translation quality without increasing inference cost.

5.7.1 Model Selection

When selecting which MT models to benchmark, we considered several aspects, including the number of model parameters, model size, inference time and reported BLEU scores. When selecting models, the first requirement was the model's ability to translate between all six selected languages in any direction. This requirement narrowed down the choices considerably and meant that smaller models were ruled out because they do not typically support all of our six languages. An alternative would have been to use pairs of smaller models, but the device does not have enough memory to support multiple language model pairs. But what is

Variant	Backend	Prep.	Precision	Quantized components
NeMo FP32	NeMo	Original	FP32	None
ONNX pre-exp FP32	ONNX	Pre-exp	FP32	None
ONNX preproc FP32	ONNX	Preproc	FP32	None
ONNX local FP32	ONNX	Local	FP32	None
Preproc INT8 dyn m+g	ONNX	Preproc	INT8 dyn.	MatMul, Gemm
Preproc INT8 dyn enc m+g	ONNX	Preproc	INT8 dyn.	Encoder MatMul, Gemm
Local INT8 dyn enc m+g	ONNX	Local	INT8 dyn.	Encoder MatMul, Gemm
ONNX pre-exp INT8	ONNX	Pre-exp	INT8	Vendor-provided INT8 graph
Preproc INT8 dyn enc gen	ONNX	Preproc	INT8 dyn.	Encoder, general
Preproc INT8 dyn enc m+g+c	ONNX	Preproc	INT8 dyn.	Encoder MatMul, Gemm, Conv
Local INT8 dyn enc m+g+c	ONNX	Local	INT8 dyn.	Encoder MatMul, Gemm, Conv
Preproc INT8 dyn m+g+c	ONNX	Preproc	INT8 dyn.	MatMul, Gemm, Conv
Preproc INT8 dyn gen	ONNX	Preproc	INT8 dyn.	General

Table 5.2: Parakeet deployment variants evaluated in the focused optimisation study.

Note. Pre-exp denotes the vendor-provided pre-exported ONNX model; Preproc denotes the ONNX model after graph preprocessing prior to quantization; Local denotes the locally exported ONNX bundle; dyn denotes dynamic post-training quantization; enc denotes encoder-only quantization; gen denotes broader supported-operator quantization; m+g denotes MatMul and Gemm quantization; and m+g+c additionally includes Conv.

Model	Params [M]	Size [GB]	DType	GFLOPs
opus-mt-mul-en-mul	155.01	0.62	FP32	18.83
M2M100 418M	483.91	1.94	FP32	137.87
mBART-50	610.88	2.44	FP32	179.93
NLLB-200 600M	615.07	2.46	FP32	214.41

Table 5.3: Characteristics of Evaluated MT Models

possible given the constraints is to use an English centric pivot strategy. This requires two models, one that translates into English and one that translates from English into the target language. With these considerations in mind, we selected the four models listed in Table 5.3. The models are described in detail in Chapter 2.

5.7.2 Baseline comparison of pre-trained models

We evaluated all four models on our dataset described in Chapter 4. Each model translates every sentence to all six selected languages, resulting in over 30,000 translations per model. This means that we get 30 individual BLEU scores for each model, one for each directed language pair. Because our primary focus is model performance, the best-performing model was selected for further fine-tuning, without explicitly optimizing for inference speed.

5.7.3 Finetuning

Setup

Based on benchmark results and reported performance, the NLLB-200 (600M distilled) model was selected for further fine-tuning. During fine-tuning, the focus was on addressing weaknesses in model performance as measured by BLEU scores. To fine-tune our model we will utilize LoRA as our PEFT method. For all fine-tunes we target the attention projection layers (query, key, value, output). The LoRA hyper-parameters were the same for all initial fine-tunes, the rank (r) was set to 16 and the scaling coefficient (α) was set to 32, with a dropout of 0.1. AdamW was used as the optimizer, with a learning rate of 1×10^{-4} and a weight decay of 0.01. We load the model with FP16 weights to reduce memory consumption and speed up training. We utilize the transformers library from Hugging Face to make fine-tuning easier.

Single-Pair fine-tune

Our initial fine-tuning focuses on Swedish-to-French translation. This pair is selected because it was among the worst-performing language pairs in our benchmark. We use the TED2020 dataset, but only use the Swedish to French part of the dataset. Since only one of the 30 language pairs is used, we select 12,000 sentence pairs. We take out 10% of the sentences to create a validation set, leaving us with 10,800 sentences to train on. The training configuration has a batch size of 64 and utilize a linear warm up for the first 100 steps.

Sequential two-pair fine-tune

After our initial fine-tune, we perform a sequential fine-tune using our initial fine-tuned model. We first evaluate the single-pair fine-tuned model and choose the worst performing pair, which in this case was Dutch to German. Instead of starting from the base NLLB-200 model, we initialize training from the single-pair model and unfreeze the LoRA matrices for further training. We use 12,000 sentences from the dataset and apply a train-validation split. We keep the hyperparameters from our single-pair fine-tune.

Multi-Pair fine-tune

After our sequential fine-tuning we perform a larger fine-tuning on 12 language pairs. Rather than training sequentially which introduces the risk of catastrophic forgetting we now train on all of the language pairs at the same time. The 12 selected language pairs include both the worst-performing pairs and pairs representing under-represented languages in the dataset. We focus specifically on German, as it has the lowest mean BLEU score both as source and as target language. This design leverages the model's strong existing cross-lingual capabilities. We use the TED2020 dataset and select 30,000 sentences per language pair. We increase the dataset size to improve learning stability across all 12 language pairs. We train for 5 epochs with a linear warm up of 500 steps.

5.7.4 Dynamic curriculum fine-tuning

Curriculum learning is a training technique that mimics human learning. The model starts by training on easy data and as the model improves, the training data becomes harder to match the model's new level of understanding (Bengio et al., 2009). A standard curriculum learning framework involves a difficulty measurer to score individual training samples and a pacing strategy to dictate how more difficult data is introduced over time. This pacing strategy typically follows a fixed schedule, such as gradually increasing difficulty based on the number of training steps. Opposed to the traditional approach, we employ an adaptive task-level curriculum. Instead of assigning difficulty at the sentence level, we consider the difficulty of entire language pairs. The pacing strategy adjusts the sampling ratio of training data over time.

Initial Binary-Threshold Approach

In our initial dynamic curriculum fine-tuning, we start with all language pairs. We now use all of the TED2020 dataset with 50,000 sentences per language pair. We first train for 2,000 steps and then evaluate the BLEU scores for each language pair. We repeat this process for 30 rounds or until all language pairs reach a predefined BLEU threshold. We set the threshold to a BLEU score of 28. When a language pair exceeds the threshold, we reduce its training data by 90%, retaining 10% of the original sentence pairs. We retain 10% of the data as a buffer set to reduce catastrophic forgetting. We train with a learning rate of 1×10^{-4} and we set the LoRA rank (r) to 16 and the scaling coefficient (α) to 32.

Continuous Weighting Function & Decay Model

To further improve the model, we redesign how language pairs are phased out during training. Using a binary threshold for graduation, where a language pair is suddenly dropped, can hurt model performance. This happens because the training data get aggressively altered between rounds, meaning that the gradient, optimizer states, and shared representations undergo a severe distributional shock. Because the AdamW optimizer relies on historical tracking, a drastic data reduction renders these tracked states unrepresentative of the new data distribution. To mitigate this issue, we introduce a continuous weighting function. In this approach the proportion of training data per language pair is dynamically adjusted based on the latest evaluation. We implement this as an exponential decay function, where the sampling weight w is computed from the BLEU score s of each language pair. If the score is below our minimum BLEU threshold, we keep all the samples. We set the minimum BLEU score (s_{min}) to 28. If the BLEU score exceeds the graduation threshold, only 1% of the samples are retained. We have set our graduation score (s_{grad}) to 45. Otherwise, the weight is computed using the exponential decay defined in (5.2), where the normalized score $t(s)$ is defined in (5.3).

$$w(s) = \begin{cases} 1.0 & \text{if } s < s_{\text{min}} \\ w_{\text{min}} & \text{if } s \geq s_{\text{grad}} \\ \max\left(w_{\text{min}},\, e^{-3 \cdot t(s)}\right) & \text{otherwise} \end{cases} \tag{5.2}$$

$$t(s) = \frac{s - s_{\text{min}}}{s_{\text{grad}} - s_{\text{min}}} \tag{5.3}$$

5.8 Integrated Speech Translation Pipeline

After evaluating the ASR and MT components independently, we combined them into a cascaded speech translation pipeline. For each utterance, the ASR component first generated a source-language transcript which was then passed to the MT component to generated the target-language translation.

The ASR component we retained was the ONNX Runtime Parakeet variant with dynamic INT8 quantization restricted to MatMul and Gemm operators. This configuration was selected because it provided the strongest observed balance between WER, latency, and memory use in the focused deployment study. The MT component was the single-pair finetune of NLLB.

As can be seen in Figure 5.1, we distinguished between two input conditions for the MT stage in order to separate translation quality from transcription-induced error propagation. In the noisy condition, the translation model received ASR-generated transcripts, representing realistic cascaded inference. In the clean condition, the translation model received reference source-language text, which helped us isolate how much degradation arose from upstream recognition errors.

The pipeline followed a strictly sequential processing flow and was evaluated under the same conditions as the constrained 8 GiB ASR benchmark. We did not use batching, asynchronous scheduling, streaming, or other throughput-oriented optimisations, because these would obscure the per-utterance latency of a local interaction. The reported end-to-end

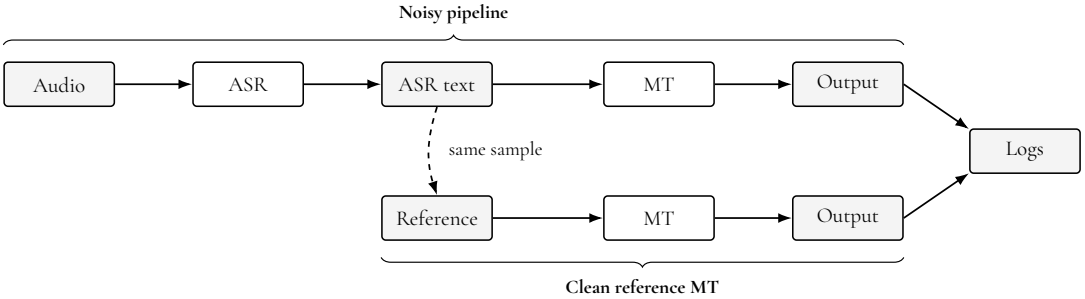

Figure 5.1: Integrated pipeline evaluation with noisy ASR-input and clean reference-input MT conditions. The noisy branch represents deployable cascaded inference, while the clean branch isolates translation quality without upstream ASR errors.

measurements therefore reflect a conservative sequential baseline for local cascaded speech translation.

Together, the methodology supports a conditional feasibility claim. The experiments do not test every possible local speech translation architecture, hardware platform, or language set. Instead, they measure how selected ASR and MT components behave under controlled CPU and memory limits, and whether their combined accuracy, latency, and resource use are compatible with local multilingual speech translation in the evaluated setting.

Chapter 6
Results

The results show how feasibility changes as the system moves from isolated components to the integrated speech translation pipeline. The ASR experiments show that accurate real-time recognition is possible under the evaluated 8 GiB, four-core setting when the model and runtime backend are selected carefully. The complete pipeline, however, remains limited by MT latency and by the quality loss introduced when translation is performed from ASR generated transcripts. This chapter follows our staged evaluation design, moving from ASR to MT, then the pipeline, and finally the system-level feasibility.

6.1 Automatic Speech Recognition

The ASR experiments evaluate whether multilingual speech recognition can be performed locally with sufficient accuracy and latency to serve as the upstream component of the cascaded ASR-MT pipeline. First, we compare pretrained ASR models under accuracy-oriented conditions, then we evaluate the most relevant candidates under constrained CPU and memory settings, and finally we report the focused Parakeet deployment study. The purpose is not only to identify the lowest-WER candidate, but to determine which ASR configuration offers the strongest deployment trade-off between recognition quality, latency, and memory use.

6.1.1 Comparison of Pretrained ASR Models

Table 6.1 shows the initial WER comparison for the models across the six-language evaluation corpus. The results show a clear hierarchy when it comes to transcription accuracy. Within the Whisper family WER decreases as model size increases. Whisper-tiny has the highest corpus WER at 34.29%, while Whisper-medium reduces it to 7.11%. This indicates that additional model capacity improves transcription quality on the evaluated corpus, although the

later constrained experiments show that this improvement must be weighed against latency and memory cost.

Parakeet achieves the strongest transcription accuracy, with a corpus WER of 3.58% and the lowest WER for every evaluated language. Its largest remaining error rate is observed for Swedish, at 8.31%, but this is still below the Swedish WER of all evaluated Whisper variants. Even though they are approximately the same size, Parakeet reduces corpus WER by 3.52 percentage points in comparison with Whisper-medium, making it the strongest accuracy candidate prior to introducing deployment constraints.

The per-language results also show that the smaller Whisper models are uneven across the selected languages. Dutch and Swedish are particularly difficult for Whisper-tiny and Whisper-base, while English and Spanish generally produce lower WER. This matters for the pipeline because ASR errors are not only recognition errors, they also become source-side noise for MT.

Model	Dutch	English	French	German	Spanish	Swedish	Corpus
Whisper-tiny	56.25	12.03	31.91	29.29	25.76	52.56	34.29
Whisper-base	37.89	8.87	18.16	19.77	15.13	36.92	22.43
Whisper-small	20.50	6.68	8.20	10.57	6.53	18.58	11.59
Whisper-medium	10.98	5.52	5.01	6.72	4.25	11.12	7.11
Parakeet	**3.57**	**3.09**	**2.39**	**2.99**	**1.97**	**8.31**	**3.58**

Table 6.1: Word error rate (%) of pretrained ASR models across the six evaluation languages. Lower values indicate better transcription accuracy.

Overall, the initial comparison shows that Parakeet provides the best transcription accuracy across the multilingual evaluation set. We opted to exclude Whisper-base from later constrained ASR comparison because it occupied an intermediate point between Whisper-tiny and Whisper-small, while the remaining models provided clearer trade-off points across accuracy, latency, and memory use.

6.1.2 Performance under 8 GiB Memory Constraint

Tables 6.2 and 6.3 report the constrained ASR benchmark under a fixed deployment budget of four CPU cores and an 8 GiB container memory limit. All evaluated models remained operational under this configuration, but their accuracy, latency, and resource profiles differed significantly.

Parakeet achieved the strongest overall operating point, combining the lowest corpus WER, 3.58%, with the lowest median RTF, 0.098, and the lowest 95th percentile RTF, 0.124. This indicates not only that the transcription is faster than real-time, but also that its latency distribution is consistently tight across the evaluated samples. Whisper-tiny has a similar median RTF of 0.101, but its WER is 36%, making it unsuitable as the main recogniser for a translation pipeline where transcription errors propagate downstream.

Whisper-small occupies an intermediate position, with lower WER than Whisper-tiny but substantially higher latency. Whisper-medium improves recognition quality further, but its median RTF of 1.268 and 95th percentile RTF of 2.004 show that it fails to sustain real-time behaviour in the evaluated environment. This illustrates why accuracy alone is insufficient for model selection as Whisper-medium is more accurate than the smaller Whisper variants, but its latency makes it a weaker deployment candidate.

Model	WER [%]	RTF_{50}	RTF_{95}	t_{50} [s]	t_{95} [s]
Whisper-tiny	36.00	0.101	0.147	1.04	1.62
Whisper-small	16.96	0.435	0.687	4.47	5.94
Whisper-medium	13.28	1.268	2.004	12.99	16.55
Parakeet	**3.58**	**0.098**	**0.124**	**1.02**	**1.60**

Table 6.2: Recognition accuracy and latency of the evaluated ASR models under constrained inference with four CPU cores and an 8 GiB container memory limit. Lower values indicate better performance.

The increase in WER for the Whisper-models relative to the initial baseline comparison should also be interpreted in light of the decoding configuration. In the initial accuracy-oriented benchmark, the spoken language was explicitly specified to the model, whereas in the constrained evaluation this information was not provided. The resulting WER increase is therefore the combined effect of deployment-oriented inference and model limitations in automatic language identification, rather than a pure consequence of the resource constraints.

Qualitative inspection of the transcriptions further revealed that both Whisper-tiny and Whisper-small were more prone to language misclassification, particularly when processing English speech with non-native accents. In such cases, the models occasionally entered unstable decoding behaviour, producing repetitive or looping outputs that did not correspond to the input speech. These inference loops were often triggered following incorrect language identification and led to severe transcription errors, disproportionately increasing WER and, in some cases, latency.

The resource measurements in Table 6.3 show the opposite side of the trade-off. Parakeet has the highest memory demand, with mean RSS of 6.19 GiB and peak RSS of 7.19 GiB. This remains within the 8 GiB limit, but leaves very limited memory headroom for other system components. Whisper-tiny, Whisper-small, and Whisper-medium require less memory but none of them match Parakeet's combined latency and accuracy profile.

CPU utilization was high for all models, indicating substantial use of the available four-core CPU budget. Whisper-medium and Whisper-small had the highest mean CPU utilization, at 381.80% and 372.92%, respectively. Parakeet used 360.87% mean CPU and a 95th percentile of 382.08%, while Whisper-tiny used the least CPU on average.

Taken together, the 8 GiB constrained benchmark shows that Parakeet was the only evaluated model that combined low WER with consistent real-time inference. Its main limitation was its memory use, which was considerably larger than that of the Whisper models.

Model	Process RSS [GiB]		CPU utilization [%]	
	Mean	Peak	Mean	P95
Whisper-tiny	**1.75**	**2.78**	**325.61**	**341.34**
Whisper-small	2.79	3.84	372.92	397.55
Whisper-medium	4.71	5.90	381.80	398.68
Parakeet	6.19	7.19	360.87	382.08

Table 6.3: Resource usage of the evaluated ASR models under constrained inference with four CPU cores and an 8 GiB container memory limit.

6.1.3 ASR Feasibility under a 4 GiB Memory Budget

To further assess deployment feasibility, the constrained ASR experiments were repeated under a stricter memory constraint of 4 GiB while maintaining the same four-core CPU allocation. This experiment was intended to determine which models remained viable when the available memory budget was reduced by half.

Tables 6.4 and 6.5 show how memory, feasibility and recognition quality are constantly in tension. The models that fit within the 4 GiB are the smallest Whisper variants, but these are also the weakest recognizers. Whisper-tiny remains faster than real time, with a median RTF of 0.116, but its WER remains 36%. Whisper-small improves WER to 16.96%, with a median RTF of 0.495, while also leaving little memory headroom with peak RSS of 3.82 GiB. Whisper-medium and Parakeet were infeasible under the 4 GiB limit in their evaluated forms, showing how a strong candidate cannot simply be transferred to a stricter memory budget without additional optimisation or model changes.

Model	WER [%]	RTF_{50}	RTF_{95}	t_{50} [s]	t_{95} [s]
Whisper-tiny	36.00	**0.116**	**0.172**	**1.20**	**1.88**
Whisper-small	**16.96**	0.495	0.760	5.01	6.82
Whisper-medium	*Infeasible*				
Parakeet	*Infeasible*				

Table 6.4: Recognition accuracy and latency characteristics of the evaluated ASR models under constrained inference using 4 CPU cores and an 4 GiB container memory limit. Lower values are better.

6.1.4 Impact of Backend and Quantization Strategy on Parakeet Deployment

The constrained ASR evaluation showed that Parakeet achieved the best accuracy-latency trade-off under the 8 GiB memory limit, but it also had the highest memory footprint among

Model	Process RSS [GiB]		CPU utilization [%]	
	Mean	Peak	Mean	P95
Whisper-tiny	**1.71**	**2.69**	**315.61**	**334.95**
Whisper-small	2.80	3.82	360.07	395.18
Whisper-medium	*Infeasible*			
Parakeet	*Infeasible*			

Table 6.5: Resource usage of the evaluated ASR models under constrained inference with four CPU cores and a 4 GiB container memory limit.

the evaluated models. In order to understand how to improve its deployment characteristics, we conducted a focused deployment study to evaluate whether backend selection, exported graph preparation, and dynamic INT8 quantization could improve its runtime and memory behaviour without recognition degradation.

The results in Tables 6.6 and 6.7 show that execution backend has a large effect before quantization is even applied. Switching from native NeMo to ONNX based execution had a significant effect on runtime behaviour and improved latency while nearly preserving WER. The FP32 ONNX variant all reported WER values of 3.59%, compared with 3.58% for NeMo FP32. At the same time, median RTF decreased from 0.098 to 0.061-0.062 for the FP32 ONNX variants. This indicates that switching from NeMo to ONNX based execution improved ASR latency without degrading recognition quality in a meaningful way within the evaluation setup.

The strongest overall operating point without clear WER degradation was obtained by selective dynamic INT8 quantization of the MatMul and Gemm operators. The preprocessed INT8 dynamic MatMul/Gemm variant retained a WER of 3.59%, matching the FP32 ONNX variants, while reducing median RTF to 0.031. Its 95th percentile RTF was 0.037, indicating a tight, stable, and significantly faster-than-real-time latency distribution

By contrast, broader quantization did not further improve the trade-off. The vendor-provided pre-exported INT8 variant reduced latency compared with FP32 ONNX execution, but its WER increased to 5.41%. Variants that quantized convolutional operators or used broader general quantization reached WER values around 6.37%-6.42%. These variants remained faster than the FP32 variants, but their recognition degradation was worse than that of the selectively quantized MatMul/Gemm variants. These results indicate that, for this exported Parakeet configuration, selective quantization of dense operators gave a better accuracy-latency trade-off than broader quantization.

Table 6.7 shows that the backend change also reduced memory use. Switching to ONNX reduced mean RSS to approximately 3.91-4.04 GiB and peak RSS to approximately 5.04-5.12 GiB. Dynamic INT8 quantization reduced memory further, where the selective MatMul/Gemm variants had mean RSS values around 2.51-2.57 GiB and peak values around 3.59-3.65 GiB. Broader quantization produced the lowest memory footprint, but this came with higher WER

To summarise, the best deployment choice depends on the objective. If minimum memory is the priority, broader INT8 quantization gives the smallest footprint. If the goal is to

Variant	WER [%]	RTF_{50}	RTF_{95}	t_{50} [s]	t_{95} [s]
NeMo FP32	**3.58**	0.098	0.124	1.02	1.60
ONNX pre-exp FP32	3.59	0.062	0.071	0.64	1.04
ONNX preproc FP32	3.59	0.061	0.074	0.64	1.05
ONNX local FP32	3.59	0.061	0.071	0.63	1.03
Preproc INT8 dyn m+g	3.59	**0.031**	**0.037**	**0.32**	**0.51**
Preproc INT8 dyn enc m+g	3.61	0.033	0.039	0.34	0.53
Local INT8 dyn enc m+g	3.61	0.033	0.040	0.34	0.54
ONNX pre-exp INT8	5.41	0.035	0.040	0.36	0.58
Preproc INT8 dyn enc gen	6.37	0.051	0.057	0.53	0.85
Preproc INT8 dyn enc m+g+c	6.37	0.051	0.056	0.53	0.86
Local INT8 dyn enc m+g+c	6.37	0.050	0.056	0.52	0.84
Preproc INT8 dyn m+g+c	6.41	0.050	0.055	0.51	0.83
Preproc INT8 dyn gen	6.42	0.047	0.052	0.49	0.80

Table 6.6: Transcription accuracy and latency for the evaluated Parakeet variants. Lower values indicate better performance.

Variant	Process RSS [GiB]		CPU utilization [%]	
	Mean	Peak	Mean	P95
NeMo FP32	6.19	7.19	360.9	382.1
ONNX pre-exp FP32	3.91	5.04	374.9	392.0
ONNX preproc FP32	4.02	5.08	380.6	391.8
ONNX local FP32	4.04	5.12	380.2	392.0
Preproc INT8 dyn m+g	2.51	3.59	367.9	379.4
Preproc INT8 dyn enc m+g	2.54	3.62	366.1	378.8
Local INT8 dyn enc m+g	2.57	3.65	367.3	380.2
ONNX pre-exp INT8	2.71	3.80	323.6	343.4
Preproc INT8 dyn enc gen	2.34	3.41	327.6	343.6
Preproc INT8 dyn enc m+g+c	2.43	3.41	325.4	341.9
Local INT8 dyn enc m+g+c	2.37	3.44	326.8	343.5
Preproc INT8 dyn m+g+c	2.31	3.38	324.9	341.3
Preproc INT8 dyn gen	2.27	3.35	325.8	341.9

Table 6.7: Resource usage for the evaluated Parakeet variants with four CPU cores and an 8 GiB container memory limit.

Model	Mean Score	Model Extremes	
		Best Pair	Weakest Pair
mBART	13.72	39.41	0.52
OPUS-MT (mul-mul)	21.43	42.37	5.50
M2M-100	22.77	38.85	12.79
NLLB-600M	**35.88**	**61.38**	**15.04**

Table 6.8: Comparative evaluation of translation quality (BLEU) across four multilingual models.

preserve transcription quality while improving runtime and memory use, selective dynamic INT8 quantization of MatMul and Gemm operators provides the strongest observed trade-off. This latter configuration is the most suitable ASR candidate for our pipeline because it substantially reduces latency and memory use while preserving the WER of the FP32 ONNX baseline.

6.1.5 ASR Result Summary

The ASR results show that local real-time recognition is feasible under the evaluated 8 GiB and four-core constraint, but only with the right model and runtime configuration. Whisper-tiny is fast but too inaccurate for the pipeline while Whisper-medium improves accuracy but fails to meet the real-time latency requirement. Parakeet provides the strongest baseline trade-off, and ONNX Runtime with selective dynamic INT8 MatMul/Gemm quantization further improves latency and memory use without meaningful WER degradation.

The 4 GiB experiment bounds this conclusion. Under that stricter budget, the strongest ASR configuration is infeasible in its evaluated form. The feasibility of the ASR model therefore applies to the evaluated 8 GiB setup, not to all embedded memory budgets.

6.2 Machine Translation

6.2.1 Comparison of pretrained models

The results of the evaluation of the pre-trained models can be seen in Table 6.8. The evaluation used the FLEURS dataset, and translation performance was measured using BLEU scores. The evaluation was done across 6 languages resulting in 30 language pairs being evaluated.

As shown in Table 6.8, there are substantial performance differences between the models. The worst performing model is mBART, which yields a mean BLEU score of 13.72, indicating poor translation quality across several language pairs. However, the low mean is primarily driven by the high variance in model performance across different language pairs, with several scores falling below 1.0. This deficiency is likely due to insufficient training on low resource language pairs. The mBART model is designed to be fine-tuned for specific language

Language	Source Performance (From) Avg. Translation BLEU	Target Performance (To) Avg. Translation BLEU
Dutch	35.06	37.86
English	49.18	38.02
French	37.91	37.97
German	26.62	24.43
Spanish	37.56	44.84
Swedish	28.94	32.13

Table 6.9: Detailed cross-lingual performance for NLLB-200 (600M). Scores represent the average BLEU for all translation directions.

pairs. Fine-tuning may therefore have improved performance substantially for low resource languages. The opus-mt model and M2M-100 have similar mean values of 21.43 and 22.77 respectively. But the opus model has a much higher variance, it is especially good at translating into English and Spanish, where it has a mean BLEU score of 32.72 and 35.04. This contrasts with M2M-100, which exhibits relatively low variance and more consistent performance across language pairs. The strong English BLEU scores in opus-mt are likely due to English acting as the pivot language. In these cases, only one of the two translation models is used, which reduces the risk of cascaded degradation. The model also performs strongly for Spanish, likely because Spanish is a high resource language. M2M-100 has generally promising results, it is best at translating to English, which is not surprising as it has mostly been trained on English translations. What is surprising, is the poor performance on translating from Spanish to one of the other selected languages. Overall, NLLB-200 outperforms the alternative baselines by a substantial margin, achieving a mean BLEU score of 35.88. After having compared all 4 models, we decided to continue to look at NLLB-200 as its results were much better than the other models.

6.2.2 NLLB-200

A deeper analysis of the NLLB-200 results reveals several language-specific trends. Table 6.9 indicates that the model struggles with the German language in both translation directions. For Swedish, the model performs worse when translating from the language, while performance improves slightly when Swedish is the target language. The best performance is achieved on Spanish and English, which is expected because they are both high-resource languages. The model shows strong performance for the Dutch language. The French language has very stable scores, and shows the model's ability to translate both from and to it.

When we look at the Figure 6.1 detailing the BLEU scores across specific translation directions, several problematic language pairs become apparent. Three translation directions achieve notably low scores: German to French with a score of 15.04, Swedish to French with 16.00, and Swedish to German with 16.99. Generally, we see low scores for translating to German, which we also observed when analysing the mean values in Table 6.9. The heatmap

Figure 6.1: BLEU score heat map for the NLLB-200 (600M) model across a 6x6 language matrix.

allows us to isolate the exact directions that require optimization during the fine-tuning phase.

To evaluate the computational latency and throughput of NLLB, we executed the model in a Docker container with access to four CPU cores and 8 GiB of memory. When we look at Table 6.10, we can see that it runs faster than real-time. The results across the languages are quite similar, except for French and German which are a bit slower. When considering TPS, performance is nearly identical across all languages. RTF is a bit of a misleading metric for MT as the audio length does not exactly reflect the number of tokens. For example, a 10 second audio segment may contain only a small number of tokens. It does however give an indication on how MT compares to ASR in our cascaded pipeline. When analysing Table 6.11, the resource metrics show that NLLB-200 demands nearly all of the available deployment budget. The mean CPU utilization is 339.26%, which is very close to the maximum of 400%. Meaning that almost all cores are working at maximum capacity. Memory usage peaks at 6.08 GiB and averages 4.3 GiB, indicating that the container has plenty of memory left.

6.2.3 Fine-tune

Single pair

We begin by fine-tuning one of the worst-performing language pairs. This run is conducted to establish an understanding of how much a specific pair can be optimized and to observe whether other language pairs begin to regress. We choose to fine-tune on Swedish to French translations as it is one of the worst pairs, and also because we have some linguistic knowledge of the two languages. Figure 6.2 shows the results of the fine-tuning experiment. The overall performance gains are substantial, yielding a mean BLEU score of 38.9. This corresponds to

Target	RTF_{50}	RTF_{95}	t_{50} [s]	t_{95} [s]	TPS_{gen}
Dutch	0.575	0.812	5.99	10.05	4.882
English	0.506	0.746	5.32	8.65	4.759
French	0.655	0.961	6.61	11.05	4.965
German	0.627	0.943	6.39	10.77	4.885
Spanish	0.570	0.836	5.81	9.79	4.887
Swedish	0.562	0.816	5.66	9.29	4.830
Total	0.579	0.871	5.94	10.04	4.872

Table 6.10: Latency characteristics per target language for NLLB.

Metric	Value
Mean CPU utilization	339.26%
P95 CPU utilization	351.14%
Mean RSS	4.3 GiB
Peak RSS	6.08 GiB
Container CPU limit	4
Container memory limit	8 GiB

Table 6.11: Resource usage of NLLB pipeline during the measured benchmark window.

Fine-tune Configuration	Avg. BLEU	Best Score	Worst Score	Standard Deviation
Baseline (NLLB-200)	35.88	**61.36**	15.04	12.54
1. Single Pair	38.90	57.59	**21.65**	**9.81**
2. Sequential Two-Pair	35.12	57.49	13.48	10.44
3. Multi-Pair (12 pairs)	38.08	60.10	10.79	13.75
4. Dynamic Hard Cutoff	37.58	60.25	9.65	12.39
5. Dynamic Decay Function	**39.07**	60.50	16.57	12.08

Table 6.12: Global performance comparison across all five fine-tuning configurations.

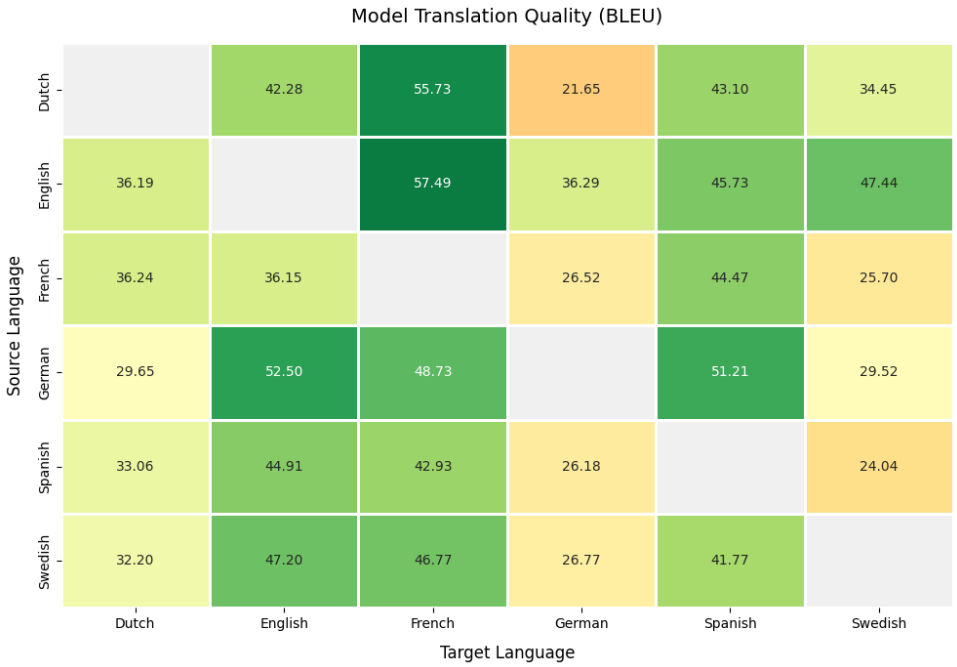

Figure 6.2: BLEU score heat map for the NLLB-200 (600M) single-pair fine-tune.

an increase of 3.02 BLEU points compared to the baseline model, indicating a strong effect of fine-tuning. The targeted translation direction, Swedish-to-French, increases from 16.00 to 46.77 BLEU. The fine-tune has also improved other French translations, for example German to French is up from 15.04 to 48.73. Some language pairs regress, for example Swedish-to-Dutch decreases from 40.13 to 32.20.

Sequential two-pair

We then performed our sequential fine-tuning, which proved to be ineffective. The evaluation shows that the model regresses by 3.87 BLEU points compared to the single-pair fine-tuned model. The model underperforms the baseline model, indicating that sequential fine-

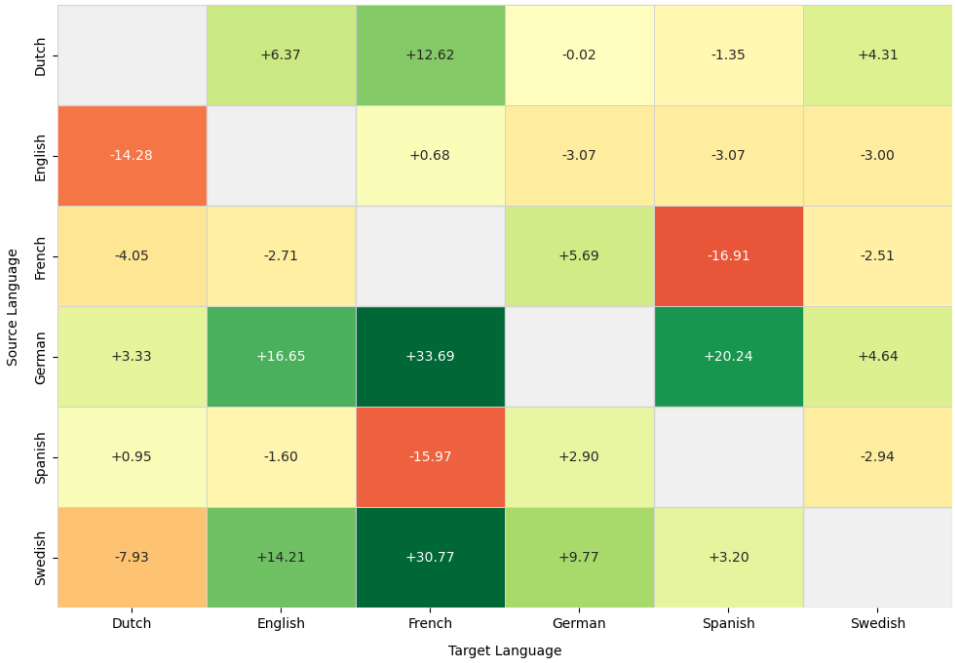

Figure 6.3: Performance delta (Δ) of the single-pair fine-tuning relative to the NLLB-200 (600M) baseline.

tuning does not improve performance for NLLB-200. This sharp decline indicates that the model suffered from catastrophic forgetting, where training on the second pair essentially overwrote the well-performing weights of the single-pair fine-tune. Without a joint training distribution, the model's parameters quickly biased toward the most recent fine-tuning targets, rendering sequential training ineffective.

Multi-pair

The multi-pair fine-tuning yields an overall mean BLEU score of 38.08. This is a significant increase compared to the baseline model's performance, but it still falls short of the single-pair fine-tune (Table 6.12). We can observe that 3 language pairs suffered from catastrophic forgetting; the most drastic degradation occurred in the Spanish-to-Dutch pair, where the BLEU score dropped from a baseline of 32.11 to 10.79. These results highlight a potential limitation of targeted dataset design. Because the 12 pairs are selected based on the weakest baseline performance, high-performing language pairs are not included in the fine-tuning dataset. During fine-tuning, performance shifts toward the selected language pairs, and excluded language pairs experience performance degradation consistent with catastrophic forgetting.

Dynamic hard cut-off

The results for our initial dynamic curriculum fine-tuning can be seen in Table 6.12. Here we see the same tendencies as in the multi-pair fine-tuning. The Spanish-to-Dutch language pair again shows degradation consistent with catastrophic forgetting. This time performance decreases from 32.11 to 12.31, representing a slight improvement over the multi-pair fine-

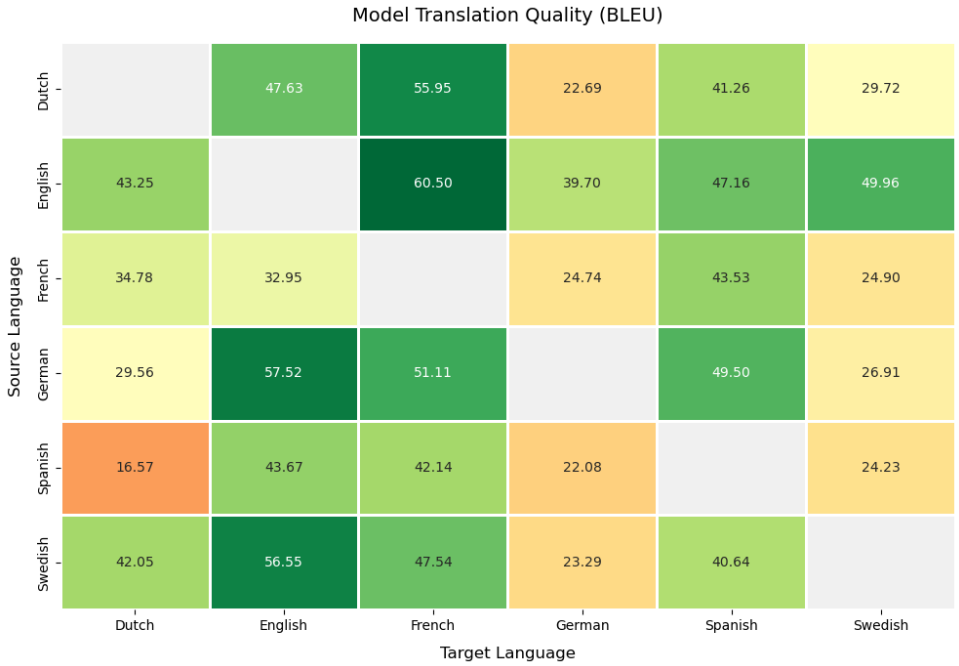

Figure 6.4: BLEU score heat map for the NLLB-200 (600M) dynamic fine-tune with decay function.

tune but remaining considerably below our baseline. When looking at the training logs, we observe that for the validation set during training, the model achieved a BLEU score of 24.46. This highlights a significant performance gap between the validation metrics recorded during the training loop and the final evaluation results on this specific pair. Despite reasonable performance during training, the final evaluation collapsed. Although some language pairs degrade, the overall results show a moderate improvement. The results are more uniform and show much lower variance than the multi-pair fine-tune. This is likely due to the training set being more balanced, which prevents the model from excelling only in certain language pairs at the expense of others.

Dynamic decay function

The dynamic decay fine-tuning achieves the highest overall performance, with a mean BLEU score of 39.07. This is 1.49 BLEU points higher than the hard cut-off approach. Compared to the single pair fine-tune, the model performs better but has a significantly higher standard deviation. The standard deviation increases from 9.81 to 12.08 compared to the single-pair fine-tune. This is a significant jump and means the model is more unstable than the single pair fine-tune. The Spanish-to-Dutch language pair, for example decreases from 32.11 down to 16.57. The standard deviation is however lower than for our baseline and hard cut-off fine-tunes. These results suggest that gradually phasing out training data for well-performing language pairs improves performance. Limiting the severe distributional shocks and optimizer confusion observed in earlier hard cut-off fine-tune.

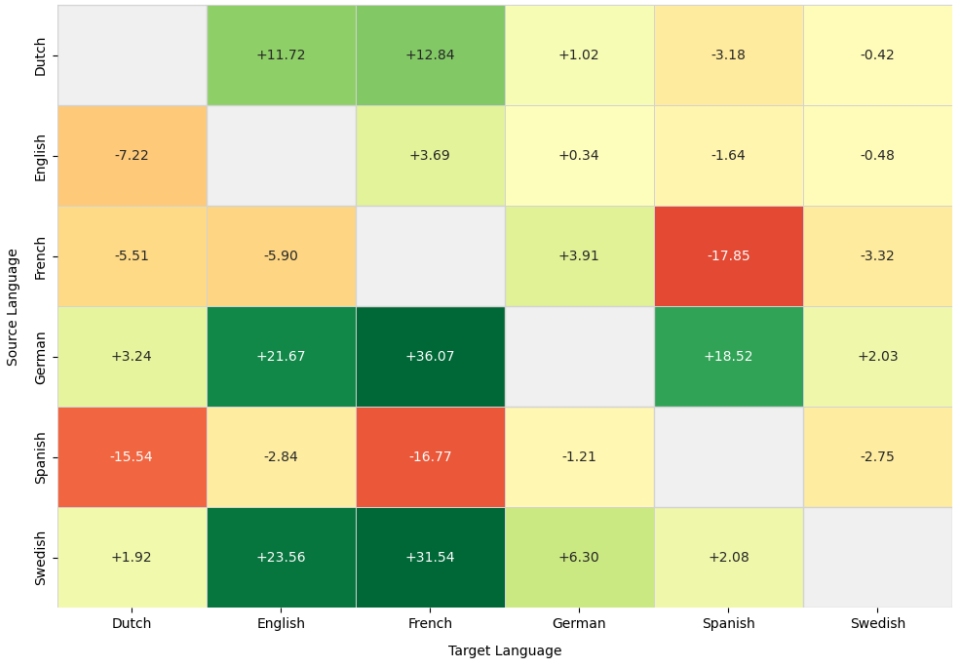

Figure 6.5: Performance delta (Δ) of the dynamic curriculum fine-tuning relative to the NLLB-200 (600M) baseline.

Condition	BLEU
Clean Input	38.03
Noisy Input	34.72
Difference	-3.31

Table 6.13: End-to-end pipeline quality under clean and noisy translation-input conditions.

6.3 Integrated Speech Translation Pipeline

6.3.1 Clean and Noisy Translation Quality

Table 6.13 compares translation quality under clean and noisy MT input conditions. In the clean condition, the MT model receives the reference source-language transcript, while the noisy condition provides it with the ASR-generated transcript. The differences between these two conditions estimate the effect of recognition errors on downstream translation quality.

As described in Section 4.2, the stand-alone MT benchmark and the pipeline evaluation use different FLEURS-derived corpus variants. The stand-alone MT benchmark used the full text corpus, while the pipeline used the filtered valid-audio subset. The following results should therefore be not be interpreted as a direct re-evaluation of the stand-alone MT benchmark on the same samples.

The noisy condition reduces BLEU by 3.31 points compared with the clean reference-

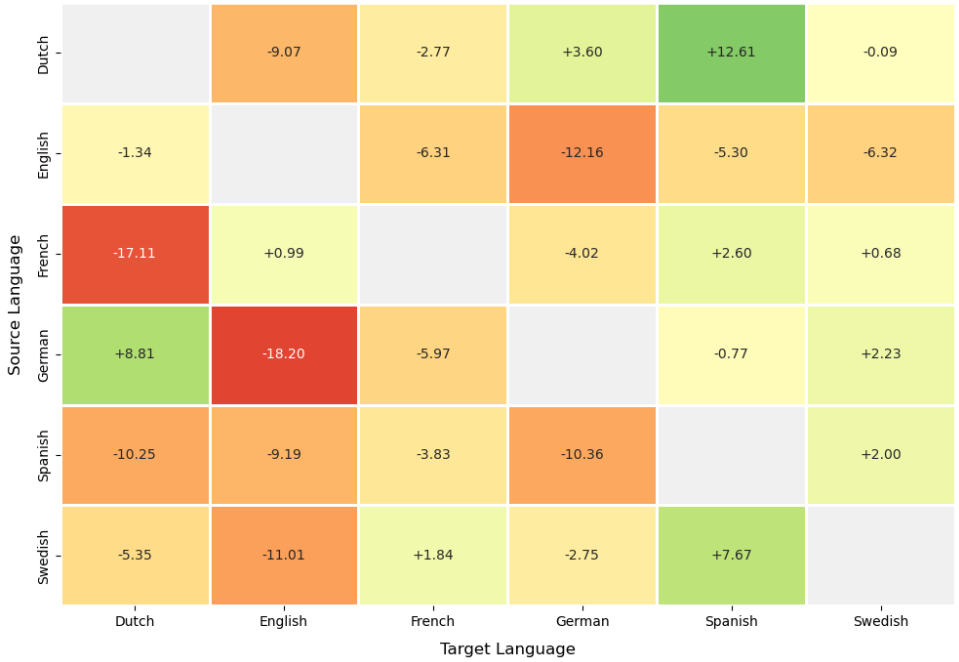

Figure 6.6: BLEU score comparison for NLLB-200 (600M) on our single-pair fine-tune using clean versus noisy input.

input translation. This confirms that recognition errors propagate into the translation stage, but the magnitude of the degradation is smaller than might be expected. This could be due to the fact that the selected ASR configuration produces relatively low WER, so the MT model often receives source text that remains close enough to the reference to preserve much of the translation quality. However, the degradation still shows that MT evaluation on clean text still overestimates the quality of the deployment pipeline.

The language-pair breakdown in Figure 6.6 shows that the degradation is not uniform. German-to-English and French-to-Dutch see notable decreases in performance, with a loss of -18.20 and -17.11, respectively. For some pairs the opposite happens, they see an increase in performance with the noisy data. Dutch-to-Spanish sees the biggest increase in performance with an increase of 12.61 compared to clean input data. The mean BLEU score for Dutch increases by 0.85, which may reflect corpus-specific characteristics or favourable overlap between the noisy ASR output and the MT model's training distribution. For the rest of the languages they see a decrease in mean BLEU score, which is expected when the input data quality decreases.

Overall, the clean/noisy comparison supports the claim that component-level MT evaluation is necessary but not sufficient. The integrated pipeline must be evaluated with ASR generated transcripts because the deployable system operates on noisy upstream output rather than reference text.

6.3.2 End-to-End Latency

Table 6.14 shows the latency and RTF for the integrated pipeline under clean and noisy MT input conditions. The ASR row corresponds to the optimised ASR configuration used in

the pipeline, the MT row corresponds to translation time, and the ASR+MT row reports the complete sequential processing chain for the noisy ASR-input condition.

The results show that the integrated pipeline remained faster than real time for the median sample, with a RTF of 0.695. However, the 95th percentile full-pipeline RTF was 1.018, slightly above the real-time threshold, thus, the system was not consistently real-time under the evaluated setup

The latency decomposition also shows that MT in its current form dominated end-to-end runtime. The ASR stage had median RTF of 0.033 and a 95th percentile of 0.039, while the MT stage had corresponding values of 0.662 and 0.982, respectively. This means that the MT stage occupied between 95 and 96% of the component latency within the pipeline. Further latency improvements for the pipeline should therefore prioritize improving the MT models latency, either through speeding up decoding, increasing model efficiency, or introducing streaming translation policies rather than focusing on accelerating the ASR component.

Stage	RTF_{50}	RTF_{95}	t_{50} [s]	t_{95} [s]
ASR	0.033	0.039	0.339	0.531
MT (Pipeline)	0.662	0.982	6.799	11.542
ASR + MT	0.695	1.018	7.150	12.026

Table 6.14: End-to-end latency and real-time factor of the integrated speech translation pipeline

6.3.3 Pipeline Resource Usage

Table 6.15 reports the system-level resource usage for the full pipeline. The integrated pipeline used a substantial portion of the available CPU budget, with mean CPU utilization of 327.16% and 95th percentile utilization of 341.75%. This indicates that the pipeline actively used the allocated four-core budget, but did not saturate it continuously on average.

Memory usage remained within the 8 GiB container limit, with a mean process RSS of 5.67 GiB and a peak of 7.11 GiB. This leaves limited but sufficient headroom in the evaluated setting. As the integrated pipeline includes both ASR and MT components, it shows that memory feasibility cannot be inferred from assessing each component individually.

The resource measurements show that the integrated pipeline is feasible under the 8 GiB, four-core setting, but with very limited memory margin. Combined with the latency results, this suggests that the main deployment issue is not whether the system can run, but whether it can consistently meet real-time latency requirements while preserving translation quality.

6.4 System-Level Feasibility Summary

The results show that multilingual speech translation is technically feasible under the evaluated 8 GiB, four-core setting, but only with careful component selection and deployment-oriented optimisation. The optimised ASR stage was accurate, faster than real time, and no longer the main latency bottleneck. This result depended not only on the pretrained Parakeet

Metric	Value
Mean CPU utilization	327.16%
P95 CPU utilization	341.75%
Mean process RSS	5.67 GiB
Peak process RSS	7.11 GiB
Container CPU limit	400%
Container memory limit	8.00 GiB

Table 6.15: Resource usage of the integrated ASR–MT pipeline during the measured benchmark window.

model, but also on the runtime backend and quantization scope. ONNX runtime execution and selective dynamic INT8 quantization of MatMul and Gemm operators gave the best observed trade-off between WER, latency, and memory use.

The integrated pipeline was instead limited mainly by MT. Although the full system remained faster than real time for the median sample, its 95th percentile RTF exceeded 1.0. The pipeline therefore did not provide consistently real-time behaviour in the evaluated setup. MT accounted for most of the measured component latency, while ASR contributed only a small fraction of the end-to-end runtime. Translation decreased when the MT model received ASR-generated transcripts rather than clean reference text, showing that clean text evaluation overestimates deployable pipeline quality.

Memory usage remained within the 8 GiB limit, but with limited headroom. The stricter 4 GiB ASR experiments further bounds the feasibility claim, since the strongest ASR candidate was infeasible under that memory budget in its evaluated form. The final conclusion is therefore not that local multilingual speech translation is generally solved, but that a carefully optimised cascaded pipeline can run under the evaluated 8 GiB setup while remaining limited by MT latency, memory margin, and ASR-induced translation degradation.

Chapter 7
Conclusion

7.1 Summary of Findings

This study evaluated whether a cascaded ASR-MT pipeline can support local multilingual speech translation under deployment-oriented CPU and memory limits. It covered six European languages and combined component-level experiments with an integrated end-to-end pipeline evaluation. The central question was not whether ASR or MT can perform well in isolation, but whether the complete system can preserve useful quality while meeting latency and resource requirements at the same time.

The ASR results show that accurate local recognition is feasible under the evaluated 8 GiB, four-core setting when model and runtime configuration are selected carefully. Among the evaluated ASR models, Parakeet provided the best balance between accuracy and latency. The focused deployment study further showed that ONNX Runtime execution and selective dynamic INT8 quantization of MatMul and Gemm operators substantially reduced latency and memory use while preserving recognition quality close to the FP32 ONNX baseline. These results show that ASR deployment feasibility depends on more than just the pretrained model, as runtime backend, graph preparation, and quantization scope are key factors to consider.

Among the MT models, NLLB proved to be the most capable model. It achieved a mean BLEU score of 35.88 across the 30 language pairs, which was considerably higher than the other 3 evaluated models. NLLB operated faster than real time but was slower than the quantized ASR models. It required 6.08 GiB of peak memory, leaving some memory headroom for the rest of the system. Despite NLLB's strong mean BLEU score, it had problematic language pairs, whose scores were below our threshold of 25. A BLEU score above 25 is considered good and resembles an acceptable translation quality. To improve low-performing language pairs, we applied LoRA, a PEFT method. We performed several fine-tunes, many of which suffered from catastrophic forgetting. Our initial single-pair fine-tuning run was successful and indicated that one language pair influenced performance across other pairs. Ultimately,

our most successful fine-tune in regards to the mean BLEU score was the dynamic curriculum fine-tune utilizing a continuous decay function. This model achieved a mean BLEU score of 39.07, corresponding to an improvement of 3.19 BLEU points over the baseline. The model that we decided to use for our prototype was the single-pair fine-tune. This was due to it having the least number of language pairs below our threshold of 25 BLEU points, reflected in its worst score of 21.65. Furthermore, it achieved the lowest standard deviation of 9.81, indicating the most consistent performance across all 30 language pairs.

The integrated pipeline gives the most important result. With the optimised ASR configuration and NLLB-based MT, the system could run locally within the evaluated 8 GiB, four-core setting. However, it was not consistently real time. The pipeline remained faster than real time for the median sample, but its 95th percentile RTF exceeded the real-time threshold. The latency decomposition further showed that the MT dominated end-to-end runtime, while the ASR stage contributed only a small fraction of the measured latency.

Translation quality also decreased when the MT model received ASR-generated transcripts instead of clean reference text. This confirms that ASR errors propagate into the translation stage, although the degradation was moderate with the selected ASR model.

Overall, the results support a conditional feasibility claim. Local multilingual speech translation is technically feasible under the evaluated 8 GiB, four-core setup, but the prototype is not yet production-ready. Optimised ASR was feasible and no longer the main bottleneck. The remaining limits were MT latency, limited memory headroom, and translation degradation caused by ASR-generated output.

7.2 Limitations

Our conclusions are limited to the evaluated models, language, datasets, runtime stack, and resource limits. The experiments used English, Swedish, Dutch, French, German, and Spanish, which provide a controlled multilingual evaluation setting but do not represent all languages, language families, accents, or deployment domains.

The runtime experiments were deployment-oriented but were not performed on final embedded hardware. Container CPU and memory limits made model comparisons controlled and reproducible, but they do not capture all the properties of a physical device.

The evaluated pipeline was also intentionally simplified. It used segmented utterances and a strictly sequential ASR-MT flow. It did not include streaming ASR, streaming MT, batching, asynchronous scheduling, speaker diarization, text-to-speech synthesis, user-interface integration, or user studies. Privacy and data-protection concerns motivated local processing, but the study did not perform a formal privacy, security, or legal compliance analysis.

7.3 Future Work

For the MT models, and in particular NLLB, several directions for future work remain. To make a on-device pipeline possible, the MT part would have to become considerably faster. Our results show that the MT component accounts for approximately 95% of the RTF. A future work could look at trying to decrease the runtime of the MT component, by for example looking at quantization. Although our fine-tuning approaches improved performance,

automated hyper-parameter optimization frameworks such as Optuna could be used to improve training efficiency. A future work could look at utilizing one of these frameworks to optimize the dynamic curriculum training approach used in this thesis. Another interesting future work would be to fine-tune the MT model on the output from the ASR model. This could allow the MT model to learn to compensate for errors introduced by the ASR component, potentially improving the overall translation quality of the pipeline. Another interesting future work could focus on the use of LLMs for the MT component. Future research could evaluate the feasibility of integrating highly compressed LLMs and explore whether their broader contextual awareness leads to better translation quality.

The ASR component also requires further deployment work. Although the optimised Parakeet configuration was feasible under the evaluated memory budgets, future work should examine smaller ASR models, additional quantization strategies, hardware acceleration, and memory-aware runtime configuration.

Finally, a more complete deployment prototype should include speaker diarization, streaming behaviour, text-to-speech output where relevant, and evaluation on target or near-target hardware. User-facing studies would also be needed to determine whether the measured latency and translation quality are sufficient for practical interaction. These extensions would move the work from technical feasibility towards a more complete assessment of deployable local speech translation.

AI Usage Statement

Artificial Intelligence (AI) tools were used to assist with grammar and sentence structure throughout this thesis, as well as to support code development and debugging during the implementation phase. The conceptual design, implementation, and final analysis remain entirely the original work of the authors.

References

Ahmed, Z., Seide, F., Moritz, N., Lin, J., Xie, R., Merello, S., Liu, Z., and Fuegen, C. (2025). Overcoming latency bottlenecks in on-device speech translation: A cascaded approach with alignment-based streaming mt.

Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4960–4964.

Conneau, A., Ma, M., Khanuja, S., Zhang, Y., Axelrod, V., Dalmia, S., Riesa, J., Rivera, C., and Bapna, A. (2023). Fleurs: Few-shot learning evaluation of universal representations of speech. In *2022 IEEE Spoken Language Technology Workshop (SLT)*, pages 798–805. Licensed under CC-BY-4.0.

Creative Commons (2021a). Creative commons attribution 4.0 international license. https://creativecommons.org/licenses/by/4.0/.

Creative Commons (2021b). Creative commons attribution-NonCommercial 4.0 International. https://creativecommons.org/licenses/by-nc/4.0/.

Creative Commons (2021c). Creative commons attribution-NonCommercial-NoDerivatives 4.0 international license. https://creativecommons.org/licenses/by-nc-nd/4.0/.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. (2022). Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Docker Inc. (2026a). *docker container stats*. Docker Inc.

Docker Inc. (2026b). *Resource constraints*. Docker Inc.

Esselink, B. (2000). *A Practical Guide to Localization*. J. Benjamins, Amsterdam ; Philadelphia.

European Data Protection Board (2021). Guidelines 02/2021 on virtual voice assistants. Version 2.0, adopted 7 July 2021.

Fan, A., Bhosale, S., Schwenk, H., Ma, Z., El-Kishky, A., Goyal, S., Baines, M., Celebi, O., Wenzek, G., Chaudhary, V., Goyal, N., Birch, T., Liptchinsky, V., Edunov, S., Grave, E., Auli, M., and Joulin, A. (2020). Beyond english-centric multilingual machine translation. Licensed under MIT.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. (2021). A survey of quantization methods for efficient neural network inference.

Google DeepMind (2026). Gemini 3.5 audio (live translate) model card. Technical report, Google DeepMind. Accessed: July 2026.

Google Translate Research Team, Finkelstein, M., Caswell, I., Domhan, T., Peter, J.-T., Juraska, J., Riley, P., Deutsch, D., Dilanni, C., Cherry, C., Briakou, E., Nielsen, E., Luo, J., Agrawal, S., Xu, W., Kats, E., Jaskiewicz, S., Freitag, M., and Vilar, D. (2026). TranslateGemma Technical Report.

Graves, A. (2012). Sequence transduction with recurrent neural networks.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented transformer for speech recognition.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference.

Jitsi (2025). Jiwer: Similarity measures for automatic speech recognition evaluation. Python package, version 4.0.0.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation.

Jurafsky, D. and Martin, J. H. (2026). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition. Online manuscript released January 6, 2026.

Khayrallah, H. and Koehn, P. (2018). On the impact of various types of noise on neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, page 74–83. Association for Computational Linguistics.

Kuchaiev, O., Li, J., Nguyen, H., Hrinchuk, O., Leary, R., Ginsburg, B., Kriman, S., Beliaev, S., Lavrukhin, V., Cook, J., Castonguay, P., Popova, M., Huang, J., and Cohen, J. M. (2019). Nemo: a toolkit for building ai applications using neural modules.

Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing.

Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. (2024). Awq: Activation-aware weight quantization for llm compression and acceleration.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Microsoft (2026). ONNX Runtime Documentation. https://onnxruntime.ai/docs/. Accessed: 2026-06-01.

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., and Blankevoort, T. (2021). A white paper on neural network quantization.

NVIDIA (2026). NVIDIA NeMo Speech Documentation. https://docs.nvidia.com/nemo/speech/.

ONNX (2026a). ONNX Gemm Operator Specification. https://onnx.ai/onnx/operators/onnx__Gemm.html#l-onnx-doc-gemm.

ONNX (2026b). ONNX MatMul Operator Specification. https://onnx.ai/onnx/operators/onnx__MatMul.html#l-onnx-doc-matmul.

ONNX (2026c). Open Neural Network Exchange. https://onnx.ai/.

ONNX Runtime (2025). Quantize onnx models.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In Isabelle, P., Charniak, E., and Lin, D., editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Park, T. J., Kanda, N., Dimitriadis, D., Han, K. J., Watanabe, S., and Narayanan, S. (2021). A review of speaker diarization: Recent advances with deep learning.

Post, M. (2018). A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels. Association for Computational Linguistics.

Prabhavalkar, R., Hori, T., Sainath, T. N., Schlüter, R., and Watanabe, S. (2024). End-to-end speech recognition: A survey. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 32:325–351.

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. Licensed under MIT.

Reimers, N. and Gurevych, I. (2020). Making monolingual sentence embeddings multilingual using knowledge distillation. Licensed under CC BY-NC-ND 4.0.

Rekesh, D., Koluguri, N. R., Kriman, S., Majumdar, S., Noroozi, V., Huang, H., Hrinchuk, O., Puvvada, K., Kumar, A., Balam, J., and Ginsburg, B. (2023). Fast conformer with linearly scalable attention for efficient speech recognition.

Seamless Communication, Barrault, L., Chung, Y.-A., Meglioli, M. C., Dale, D., Dong, N., Duquenne, P.-A., Elsahar, H., Gong, H., Heffernan, K., Hoffman, J., Klaiber, C., Li, P., Licht, D., Maillard, J., Rakotoarison, A., Sadagopan, K. R., Wenzek, G., Ye, E., Akula, B., Chen, P.-J., Hachem, N. E., Ellis, B., Gonzalez, G. M., Haaheim, J., Hansanti, P., Howes, R., Huang, B., Hwang, M.-J., Inaguma, H., Jain, S., Kalbassi, E., Kallet, A., Kulikov, I., Lam, J., Li, D., Ma, X., Mavlyutov, R., Peloquin, B., Ramadan, M., Ramakrishnan, A., Sun, A., Tran, K., Tran, T., Tufanov, I., Vogeti, V., Wood, C., Yang, Y., Yu, B., Andrews, P., Balioglu, C., Costa-jussà, M. R., Celebi, O., Elbayad, M., Gao, C., Guzmán, F., Kao, J., Lee, A., Mourachko, A., Pino, J., Popuri, S., Ropers, C., Saleem, S., Schwenk, H., Tomasello, P., Wang, C., Wang, J., and Wang, S. (2023). Seamlessm4t: Massively multilingual multimodal machine translation.

Sekoyan, M., Koluguri, N. R., Tadevosyan, N., Zelasko, P., Bartley, T., Karpov, N., Balam, J., and Ginsburg, B. (2025). Canary-1b-v2 & parakeet-tdt-0.6b-v3: Efficient and high-performance models for multilingual asr and ast. Licensed under CC BY-NC 4.0.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q. V., Hinton, G. E., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538.

Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

Sperber, M. and Paulik, M. (2020). Speech translation and the end-to-end promise: Taking stock of where we are. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7409–7421, Online. Association for Computational Linguistics.

Tang, Y., Tran, C., Li, X., Chen, P.-J., Goyal, N., Chaudhary, V., Gu, J., and Fan, A. (2020). Multilingual translation with extensible multilingual pretraining and finetuning. Licensed under MIT.

Team, N., Costa-jussà, M. R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., Kalbassi, E., Lam, J., Licht, D., Maillard, J., Sun, A., Wang, S., Wenzek, G., Youngblood, A., Akula, B., Barrault, L., Gonzalez, G. M., Hansanti, P., Hoffman, J., Jarrett, S., Sadagopan, K. R., Rowe, D., Spruit, S., Tran, C., Andrews, P., Ayan, N. F., Bhosale, S., Edunov, S., Fan, A., Gao, C., Goswami, V., Guzmán, F., Koehn, P., Mourachko, A., Ropers, C., Saleem, S., Schwenk, H., and Wang, J. (2022). No language left behind: Scaling human-centered machine translation. Licensed under CC BY-NC 4.0.

Team, Q. (2025). Qwen-mt: Where speed meets smart translation. https://qwen.ai/blog?id=qwen-mt. Accessed: 2026-06-24.

Tiedemann, J. and Thottingal, S. (2020). OPUS-MT – building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal. European Association for Machine Translation. Licensed under CC BY-NC 4.0.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Liu, J., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., and Zhang, M. (2024). Efficient large language models: A survey.

Xu, D., Li, T., Li, Y., Su, X., Tarkoma, S., Jiang, T., Crowcroft, J., and Hui, P. (2021). Edge intelligence: Empowering intelligence to the edge of network. *Proceedings of the IEEE*, 109(11):1778–1837.

Xu, H., Jia, F., Majumdar, S., Huang, H., Watanabe, S., and Ginsburg, B. (2023). Efficient sequence transduction by jointly predicting tokens and durations.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. (2025). Qwen3 technical report.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J.-Y., and Wen, J.-R. (2025). A survey of large language models.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-06-10

EXAMENSARBETE Breaking Language Barriers in Real Time: On-Device AI-Driven Speech Translation
STUDENTER Adrian Steene, Lucas Wittich
HANDLEDARE Pierre Nugues (LTH), Zebastian Karra-Krüger (Axis), Dan Lundgren (Axis)
EXAMINATOR Xuan-Son Vu (LTH)

En värld utan språkbarriärer

POPULÄRVETENSKAPLIG SAMMANFATTNING **Adrian Steene, Lucas Wittich**

Vi vill ta översättningen närmare den plats där samtalet sker. Många tal- och översättningstjänster bygger idag på molntjänster, men det gör systemen beroende av nätverk, externa servrar, och potentiellt känslig datadelning. I vårt arbete testar vi om en hårdvarubegränsad enhet kan ta emot tal, skriva ned det och översätta det helt lokalt.

Språkbarriärer märks mest när människor behöver förstå varandra direkt. Då räcker det inte alltid att översätta menyer eller färdiga texter i förväg. Om kommunikationen sker med tal behöver tekniken kunna följa samtalet tillräckligt snabbt för att stödja en pågående interaktion.

Idag sker mycket taligenkänning och översättning i molnet. Det gör det möjligt att använda stora och kraftfulla AI-modeller, men det innebär också att systemet blir beroende av uppkoppling, externa servrar och överföring av potentiellt känslig data. Vi ville därför undersöka om mer av arbetet kan ske där samtalet faktiskt händer.

Vårt system arbetar i två steg. Först gör taligenkänningen om ljud till text. Sedan översätter maskinöversättningen texten till ett annat språk. Uppdelningen gör idén enkel att förstå, men den ställer höga krav på helheten. Om första steget hör fel får översättningen ett sämre underlag. Om andra steget är för långsamt eller osäkert spelar det mindre roll att taligenkänningen fungerar bra.

Vi testade systemet på sex europeiska språk i en virtuell maskin med begränsad hårdvara. Resultaten visade att lokal taligenkänning kan fungera snabbare än realtid under de testade förutsättningarna, om modellen och körningen optimeras rätt. Den bästa varianten kunde vanligtvis köra långt snabbare än realtid, men hela systemet bromsades fortfarande av översättningen.

Vi testade även LoRA-baserad finjustering. I stället för att ändra i den stora originalmodellen fryser man den helt och lägger till ett litet, effektivt extra lager som tränas. Det kunde förbättra vissa svagare språkpar, men anpassningen behövde balanseras så att andra översättningar inte försämrades.

Vårt arbete visar att lokal AI-driven talöversättning är tekniskt möjlig i vissa fall, men den är ännu inte färdig för praktisk användning. För att komma vidare behövs framför allt snabbare översättning, större minnesmarginaler och fler tester i realistiska miljöer.