# Thesis Structure

## 1. Introduction

- Background on TinyML and embedded AI
- The growing demand for on-device speech intelligence
- Challenges of deploying Python-trained models on ultra-low-power microcontrollers
- Motivation for a pure C implementation
- Contributions of this work:
    - A practical conversion workflow
    - Quantization and implementation strategy
    - Evaluation on a real embedded platform?

## 2. Related Work

- Brief review of:
    - TinyML toolchains (TFLite Micro, CMSIS-NN, etc.)
    - Previous efforts in low-power speech inference
    - Manual(framework-free, resource-minimal solution) vs framework-based deployment methods

## 3. Model and Task Description

- Overview of the speech task
- Model architecture
- Dataset used for training and evaluation

## 4. Conversion Workflow from Python to C

### 4.1. Model Training and Export

- Trained PyTorch model
- Exporting weights and bias from .pth to .h file
- Weight quantization (float32 → Q7.8 or Q3.12)

### 4.2. C Inference Framework Design

- Data structures in C to represent layers and weights
- Fixed-point arithmetic choices (Q7.8, Q3.12, etc.)
- Forward-pass implementation for different layers

## 5. Experimental Setup

- Target platforms
- Evaluation metrics:

    - Accuracy
    - Flash / RAM usage
    - Inference latency
    - Power consumption

## 6. Results and Analysis

### 6.1. Accuracy Evaluation

- Quantized model vs original floating point model
- Comparison across quantization schemes (Q7.8 vs Q3.12)

### 6.2. Memory and Latency

- Inference latency on MCU (ms)
- Peak RAM and Flash usage

### 6.3. Error Sources and Trade-offs

- Sources of degradation (quantization, activations, fixed-point math)
- Trade-off: accuracy vs memory vs speed

## 7. Discussion

- Benefits and limitations of the manual C approach
- Use cases where this method is preferable over existing ML frameworks
- Reusability of the approach
- Potential for automation (e.g., scripts for weight export, model translation)

## 8. Conclusion

- Summary of key findings
- Practical implications for real-world embedded speech applications
- Future work:
    - Loop unrolling / SIMD optimization
    - Auto-code generation from Python