Efficient Embedded mmWave Human Pose Estimation

The Effects of Component Size on Model Accuracy, Latency, and Memory Usage

Robin Verver[1]

Supervisor(s): Marco Zúñiga Zamalloa[1]**, Nicole Rosi**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Robin Verver
Final project course: CSE3000 Research Project
Thesis committee: Marco Zúñiga Zamalloa, Nicole Rosi, Jana Weber

Abstract

Human-pose estimation is a technology with many applications such as for healthcare, smart-homes and new methods of human-computer interaction. However, traditional RGB camera-based systems come at significant privacy risks and can perform poorly in dark rooms. A new approach to human-pose estimation, estimating through the use of mmWave radars, could solve these problems.

MmWave creates a pointcloud of a person, rather than a direct RGB image and is therefore not affected by dark conditions, while simultaneously letting the subject stay anonymous. Current mmWave models are very accurate, on the order of centimetres, but generally too costly to run without a GPU.

In this paper we create an optimised mmWave human-pose estimation model that runs more accurately without a GPU, compared to a baseline model. We do this by analysing a baseline model to find what parts can be compressed without excessively losing accuracy.

Our improved model has an inference time of 41ms with a Mean Absolute Error (MEA) of 7.72cm on an embedded device. Compared to the baseline this model saves 85.9% latency, at the cost of 4.8% MEA accuracy.

Through finding which part can be compressed most effectively, we also gain insight in the relative importance of each component of the model. We also identify components that with further research could be improved to increase the accuracy of the model.

1 Introduction

Over the past few years, we have seen increased development towards human-pose estimation (HPE). Human-pose estimation is a broad field that describes any technology that allows detecting the skeletal pose of a person. This is usually done by estimating the coordinates of their joints, which can be connected to create the skeletal pose.

Human-pose estimation has various applications, such as motion capture, fall detection for the elderly [7], inventive new methods of human-computer interaction, and many more. This makes the field of particular interest to researchers

There are various methods of human pose estimation, such as with an RGB camera [6], Depth camera [12], or mmWave radars [10; 7; 13; 1]. RGB is the predominant approach for human-pose estimation as it is the most intuitive.

However, as the subjects are recognisable in RGB footage, it raises serious privacy concerns. The accuracy of an RGB-based pose estimation system is also negatively influenced by lighting, visual clutter and occlusions, which makes the technology harder to deploy. Instead, mmWave-based systems have recently attracted increasing attention as mmWave addresses these concerns. Unlike RGB cameras, which directly capture identifiable information about the subject, mmWave transmits certain radio waves towards the subject and analyses the reflected signals to infer their pose. This does not pose the same privacy risks as RGB camera systems, and this system works even in dark or occluded conditions.

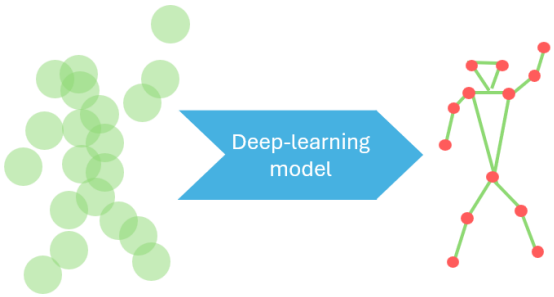

Figure 1: A deep-learning model is used to convert a pointcloud describing a person into their skeletal pose

The pose of the subject is generally inferred by using a deep-learning model. This model will, given a cloud of points describing a person, infer the X, Y, and Z coordinates of each of the joints of the subject. A simplified version of this conversion can be seen in Figure 1. These models have become increasingly accurate, with some models achieving joint position accuracy on the order of centimetres [13; 2].

However, most of the existing deep-learning models for mmWave human-pose estimation come at a major cost: Since the models are rather large, they tend to require a lot of computations to run. This means that for most practical applications the computation has to be offloaded to a dedicated GPU, which greatly increases the cost and complexity of the system. This severely limits the applications of the technology.

Some research has been done towards running compressed versions of these models without a GPU, on simpler embedded devices. This would allow for more flexibility in the design of these systems. However, this generally comes at the cost of losing significant accuracy, or having to reduce the scope of the system (such as estimating only hand gestures [10]). This loss of accuracy is something we will address in this paper.

The model used for pose estimation consists of three main components: the feature extraction, the recurrent layer, and the regression head. It is currently unknown which of these components contributes most to overall model accuracy and which components are potentially oversized for their task. This means that there are likely components of the model that can be reduced without a great loss of accuracy. If these components are reduced, this could allow the model to run more accurately on embedded devices, allowing for many more applications of the technology.

Each of the three main components consists of several neural network layers. Each of these layers has a size, which we can adjust to individually decrease or increase the computation dedicated to each component. We call these sizes *hyperparameters*. This research discusses which hyperparameters (and therefore components) of the model can be reduced

without great loss of accuracy, and finally proposes 3 new models as improvements over the baseline. One of the proposed models achieves an 85.9% reduction in latency, at the cost of 4.8% accuracy compared to the baseline.

2 Related Work

2.1 Similar Applications of mmWave

There have been many different applications of mmWave other than human pose estimation (HPE). For example, A. Ronco et al. [10] discuss hand gesture recognition from an in-ear mmWave device. This embedded device achieves 94.9% single-subject accuracy with an inference latency of 32.4ms.

Another similar application is described by H. Xue [13]. In their paper, the authors discuss mmMesh. This system uses commercial mmWave radars to estimate a body mesh of the subject standing in front of the radar. The mesh is constructed through the Skinned Multi-Person Linear model (SMPL), an algorithm to construct a human mesh consisting of 6890 vertices from 10 body and 72 pose parameters. The system first gets the data from the mmWave radars, pre-processes it, and then uses that as input for a Deep-Learning model. This model was trained to output the 82 parameters of the SMPL model. The dataset the model trained on was created specifically for mmMesh, and uses Motion Capture to establish a ground truth. The paper discusses the entire pipeline from data pre-processing to the deep-learning model that estimates the mesh parameters.

Similarly, F. Jin et al [7] created mmFall, a system that detects when an elderly person has fallen on the ground, to request a quick response from nurses. MmFall uses a dataset of normal behaviours on which a deep-learning model is trained to detect anomalies. The system also keeps track of the subjects centre of mass, which is estimated through the mmWave point cloud. If an anomaly in behaviour is detected while the centre of mass of the person quickly moves down, a fall is reported, and assistance is requested.

Most of the earlier applications of mmWave had a focus on applying the technology to solve several problems (human mesh estimation, fall detection, etc). However, little research has been done on how to create an accurate resource constrained mmWave human-pose estimation model on a microcontroller. This research will take an existing mmWave model and investigate how changes to the size of components of the model affect accuracy, memory usage and latency. This is done with the end-goal of running the model more accurately and with lower latency on a microcontroller. This makes our research question: *How do variations in the size of components affect the trade-off between accuracy, latency, and memory usage in mmWave-based pose estimation systems?*

2.2 Hyperparameter Optimisation

This paper discusses optimising the size of components (hyperparameters) compared to a baseline model. This paper does not use an algorithm to find hyperparameter configurations due to computational constraints (something that is discussed further in subsection 3.3). Nevertheless, these algorithms are interesting and would have been used given more computational resources.

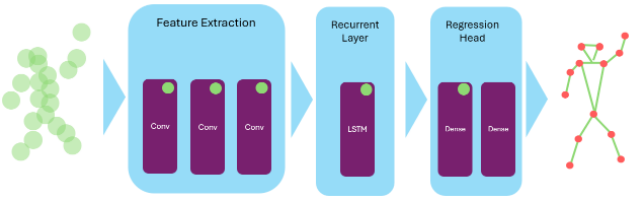

Figure 2: A simplified view of the model
with dots in the top-right indicating that the layer size can be adjusted as a hyperparameter

Most hyperparameter optimisation algorithms are based on Bayesian optimisation, more about which can be read in [11]. Bayesian optimisation trains a single configuration, evaluates it, fits the results to a Gaussian Process and uses this process to predict the performance of other configurations. It will repeatedly train configurations to improve its Gaussian Process and then use the Gaussian Process to find the next configuration expected to have high performance.

One such algorithm based on Bayesian optimisation is a Tree-structured Parzen Estimator (TPE) [3]. TPE models the probability densities of hyperparameter configurations associated with good and bad values, to select new configurations from the regions that are likely to give good performance.

Hyperband [9] is another hyperparameter optimisation algorithm. Unlike a Tree-structured Parzen Estimator, Hyperband is not based on Bayesian optimisation. Hyperband is a bandit-algorithm. It trains a set of configurations for a number of epochs. After this, it evaluates all configurations, keeping only a subset of the best configurations, and continues training only these models. This is repeated a number of times until only a single model is left.

Another approach is that which [5] takes. This paper suggests BOHB, a combined version of Hyperband and Bayesian optimisation, that has been shown to outperform both on various problems [5]. BOHB uses Hyperband to determine the number of configurations to evaluate in which budget. However, instead of using the random selection of configurations that Hyperband uses, BOHB replaces this with a model-based search.

3 Methodology

In this paper, our goal is to improve the trade-off between latency, accuracy, and memory usage by varying certain hyperparameters compared to a baseline model. In this chapter, we will describe the baseline model, the hyperparameter space we will explore, the way we will systematically find the best models in this hyperparameter space to test, and the dataset used for training.

In section 4, we will discuss the configurations we trained, the metrics used to compare to the baseline, and the results of the evaluation.

3.1 Baseline Model

The baseline model consists of three major architectural components, each of which consists of several neural-network layers. These components as well as the layers they consist of can be seen in Figure 2. The components are the feature extraction, the recurrent layer and the regression head respectively, and will be described in detail below.

Input, Output and Pre-processing We cannot directly feed the output of the mmWave radar into the neural network. Before this, the data needs to be *pre-processed*. As this thesis is mostly about optimising a pre-existing model, we will not discuss pre-processing in great detail here. In the dataset used for our training, pre-processing is already taken care of. MmMesh uses a similar pre-processing pipeline, about which more information can be found in [13].

After pre-processing, we obtain a pointcloud, where each point has 6 features: an X, Y and Z position, a range value, a velocity value and a signal intensity value. The X, Y and Z positions are relative to the radar, and the range value is the euclidean distance from the radar (simply found as $\sqrt{x^2 + y^2 + z^2}$). These points are the input to the model, generally given in batches of 32 points.

Given these features, the model predicts the X, Y, and Z positions of a number of joints. In the case of mRI dataset [1], which is the dataset used for this experiment, this is 13 joints (after pre-processing). This gives $13 \times 3 = 39$ output values. These joints describe the skeletal pose of the person described by the point cloud. We can draw lines between the positions of connected joints to visualise this skeleton, as can be seen in Figure 2.

Feature Extraction Feature extraction is the first architectural component of the model. The goal of this component is to take simple per-point information (more commonly known as *features*) and convert these to more complex *features* that apply over the entire scene. We might, for example, start with the X, Y, Z, range, velocity, and signal intensity values of a few points as its features. The feature extraction component can then 'infer' a new feature from this, for example, a value indicating how likely the user has their left arm raised. In practice, due to the nature of the training process, we cannot know what these output features actually mean, but they should describe higher-abstraction information about the scene.

This extracting of features is done through 3 *convolutional layers*. Each convolutional layer builds upon the features of the previous to output more abstract and more complex features, eventually describing enough to predict the human's pose. Compared to conventional dense layers, where each input node is connected to each output node, in a convolutional layer nodes are only connected to local nodes. The pattern of these connections is described by a kernel and shared across the entire layer. This allows us to extract features from every index, while only requiring a small number of parameters to be trained. This greatly speeds up training time, and reduces the required dataset size.

After the convolutional layers, we have extracted the required high-abstraction features from the points. We now take the average of these features of all the points to get the abstract features about the entire scene. These abstract features about the entire scene will then be processed further in the recurrent layer.

Recurrent Layer If the user had their left arm raised in the last frame, unless we get significant data showing the contrary, we can assume they likely still do. This is the main goal of the recurrent layer: we want to take previous timesteps into consideration when predicting the pose. This is done through the use of a Long Short-Term Memory (LSTM) layer.

An LSTM Layer works in three phases: the *forget gate*, the *input gate*, and the *output gate*. The LSTM remembers a set of long-term memory features between inputs to the model, which it uses to influence the next prediction. First, during the *forget gate*, depending on the input, a part of the memory is forgotten. After this, during the *input gate*, a part of the current value of the features gets put into the long term memory. Finally, at the *output gate*, the LSTM influences the input data using its memories. All this is expressed in weights, which allows the model to train itself when it should input, output or forget long-term data.

Regression Head After the recurrent layer, we have a set of abstract informations (*features*) about the scene, which is influenced by previous time-steps. The goal of the regression head is to take these features and use them to predict the X, Y, and Z positions of the person's joints. This is done through two dense layers.

The final dense layer directly outputs the coordinates we are trying to predict. This means that the size of this layer should be exactly 3 times the amount of joints we want to predict (since we are predicting an X, Y and Z value for each). In case of the mRI dataset, which has 13 joints, that leaves us with a final layer size of $13 \times 3 = 39$

3.2 Hyperparameter Space

A hyperparameter is any value that describes the architecture of the model that can be changed before training. Specifically in our case, we are interested in the dimensions (size) of specific layers of the model. We want to investigate the effect of changing these hyperparameters on the trade-off between latency, memory usage, and accuracy of the model to eventually deploy the model more accurately and with lower latency on an embedded device.

In our model there are 5 hyperparameters we can tweak. These hyperparameters can be found marked on their respective layer in Figure 2. Three of those are dimensions of the convolutional layers in the *feature extraction*, one of those is the dimension of the LSTM in the *recurrent layer*, and the final hyperparameter is the dimension of the dense layer in the *regression head*. We cannot change the dimension of the second dense layer in the regression head, since for the mRI dataset that layer has to be of size 39, as described earlier.

We vary these hyperparameters to get *hyperparameter configurations*. Each configuration is made to give us more information about the effect of the hyperparameters on accuracy, latency and memory usage. We use this information to eventually propose 3 improved models compared to the original

baseline models. The specific values for each of the configurations and the baseline can be found in Table 1.

3.3 Optimisation Techniques

There are many different approaches to exploring the hyperparameter space, each with their own advantages and disadvantages. To find the right approach, we first start by exploring the limitations of the resources for this paper. As the computational time granted to this paper is limited, excessive training of models is not realistic. Another limitation is that of current technology: Estimating memory usage and accuracy from model architecture alone is currently technologically infeasible with our resources and an area of active research. We will discuss these limitations further as we discuss the various approaches we could have taken in this paper.

Optimisation Algorithms There are a number of algorithms that can automatically search the hyperparameter space for effective models. Examples of these are Bayesian Optimisation [11], Hyperband [9], BOHB [5], and Tree Parzen Estimator [3]. These algorithms work by training several configurations, evaluating them and, using these results, predicting which other models might be valuable.

However, all of these algorithms have a common drawback: They tend to require training large numbers of models before achieving good results. Even models focused on requiring few trainings, such as Hyperband or BOHB often require at least dozens of trainings to find a good configuration [5; 9]. This is too expensive to be viable for the computational budget of this paper.

Predicting Metrics Without Training An approach to get around the budget costs might be to *predict* the expected latency, memory usage and accuracy of a model architecture rather than training and evaluating it. This would require greatly fewer computational resources, allowing many more configurations to be checked. This could be combined with one of the previously mentioned optimisation algorithms to automatically find an optimised configuration, which could then be trained and used.

Unfortunately, developing a model to estimate memory usage, latency, and accuracy is challenging due to the tight coupling between the model architecture, the implementation of the embedded deep learning library, and the specific MCU deployment target. This makes this strategy infeasible for this paper.

Manual Grid-Based Approach The final approach we will discuss is a manual grid-based approach, which is the technique that we decided to use. In this approach, instead of using an algorithm to pick the next best model to train, we set configurations at regular intervals within the hyperparameter space. For each component, we decided to explore 3 configurations: For each configuration, we change the size of the respective component to 12.5%, 25% and 200% that of the baseline. This will allow us to compare the effect each component has individually on the accuracy, latency and memory usage. We choose these sizes as we want to see the effect of great changes on each component. Smaller changes, such as 50% or 150% will likely give us less information about this.

The results from these configurations will then be used to select 3 new configurations, likely to produce better models than the baseline. We will propose these 3 configurations as improved models over the baseline. The exact configurations to be trained can be found in Table 1.

Together with one configuration for the baseline model, this makes a total of 13 configurations. Although this approach has no guarantee of finding the best model, it does achieve good results with only a few trained configurations.

The other described approaches, such as Bayesian optimisation, tend to require training more configurations to find a good model. In addition, since the results are determined by an algorithm, they tend to be less explainable. This would likely give us less useful results. Because of this, we choose the manual grid-based search approach.

3.4 Dataset

Human-pose estimation is a common application in machine learning, and therefore there are a large number of datasets that contain this type of data. Several of these also include or are dedicated to mmWave sensor data.

However, the majority of these datasets are not open-source or available. Of the eight mmWave human-pose estimation datasets identified by J. Yang. et al [14], four are open-source. The publicly available datasets identified as relevant for this work are mRI [1], Mars [2], MM-FI [14], HuPR [8] and mmBody [4].

J. Yang et al. [14] found that of the described datasets, Mars [2] is significantly smaller than the others, at 40k frames compared to 160k, 200k and 320.76k for mRI, mmBody, and MM-FI, respectively. This makes Mars a poorer candidate for the dataset used in this study.

Training time is a major constraint of this research, and therefore the dataset used should allow for training models in relatively little time. From unpublished related research, we found that this model generally trains best on 120 epochs for mRI, and 100 epochs for MM-FI. This study also found that despite requiring more epochs, the model tends to train faster using mRI rather than MM-FI. This is because mRI has fewer data frames than MM-FI, thus meaning each epoch is shorter, which outweighs the greater epoch count. This study was conducted on an RTX 4090 GPU, but we expect the relative results to be the same on the hardware used for this study. This led us to choose to use the mRI dataset.

A discussion of the diversity of the mRI dataset can be found in subsection 6.1.

4 Experiments

4.1 Explored Configurations

To find our proposed improvements over the baseline model, we train and evaluate 13 configurations. These configurations and their exact layer sizes are listed in Table 1. The first of these configurations trains the baseline model, which all other configurations will be compared to.

9 of these configurations are dedicated to exploring the relation between each component and their effect on accuracy,

Model name	Layer size				
	Feat. Extr.			Rec. Layer	Reg. Head
baseline	32	48	64	64	128
0125_feat	**4**	**6**	**8**	64	128
0250_feat	**8**	**12**	**64**	64	128
2000_feat	**64**	**96**	**128**	64	128
0125_rec	32	48	64	**8**	128
0250_rec	32	48	64	**16**	128
2000_rec	32	48	64	**128**	128
0125_head	32	48	64	64	**16**
0250_head	32	48	64	64	**32**
2000_head	32	48	64	64	**256**
impr_small	**4**	**6**	**8**	**16**	**16**
impr_medium	**8**	**12**	**16**	**16**	**16**
impr_large	**32**	**48**	**64**	**16**	**16**

Table 1: Trained configurations and their respective layer sizes. (values different from the baseline are highlighted)

latency, and memory usage. We will perform three trainings per component: each changing the size of their layers to 12.5%, 25% and 200% compared to the baseline model.

Using the results of these trainings, we will then construct three proposed improved models: A small model, which focuses on minimal latency, a large model, focusing on accuracy (while keeping a smaller latency than the baseline), and a medium model, which finds a balance between the two.

Training parameters	
Dataset	mRI
Epochs	120
Batch Size	128
Optimiser	Adam
Learning Rate	0,0005
Weight Decay	0,001
Loss Function	MSE

Table 2: The values of the training parameters while training the configurations

4.2 Training Details and Code

Training deep-learning models requires setting many parameters. Many of these parameters can be found in Table 2.

To allow the results to be recreated as best as possible, the code would ideally be published. However, this paper is part of a larger project that involves collaboration with industry. What can be published still has to be discussed, and therefore the code cannot be published at this time. We hope that the details provided here are still enough to get results with similar accuracy, memory usage, and latency metrics.

It also promotes reproducibility to set a seed before starting training to allow for reproducing the randomness of the model. However, unfortunately the authors of this paper did not know this at the time and a random seed was used. After training the models, it was considered that it was not worth retraining the models to obtain this value, as this would waste significant compute time. We hope using a different random seed will achieve similar results to what is discussed in this paper.

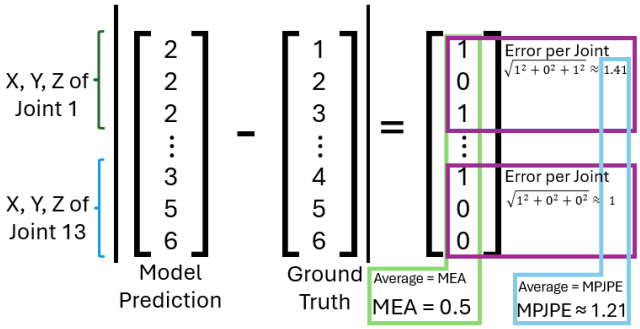

Figure 3: Visualisation of how the MAE and MPJPE metrics are calculated on a simplified example

4.3 Metrics Used to Evaluate Accuracy, Latency, and Memory Usage

In order to compare the different hyperparameter configurations quantitatively, several metrics are used. These metrics are described in this subsection.

Accuracy: MAE The Mean Absolute Error (MEA) is a metric to determine the error of each individual coordinate of each joint on average. This metric is found by subtracting the output vector of the model from the ground truth vector, and taking the average. A visualisation of how this metric is calculated is shown in Figure 3. This metric is calculated both for the quantized and unquantized versions of the model.

Accuracy: MPJPE The Mean Per Joint Position Error (MPJPE) is a metric to determine on average how far away the predicted position for a joint was from the actual position. This metric is found by first subtracting the output vector of the model from the ground truth vector. After this, the X, Y, and Z errors of each coordinate are grouped, and the absolute euclidean distance is calculated. This value is averaged over each joint, to get the MPJPE for the model. A visualisation of how this metric is calculated is shown in Figure 3. This metric is calculated both for the quantized and unquantized versions of the model.

Latency The latency is found by measuring the average time that the model takes to run on an input. This metric is measured on the embedded device.

During computation, the model weights can be stored in the SDRAM or SRAM sections of the MCU. SRAM is the fastest and smallest section of the two. However, not all models fit the SRAM section, so we have also included measurements where the model weights are stored in the SDRAM section. Using the SRAM section tends to speed up latency by roughly 2.8 times, as can be seen in the measurements in Table 6.

Memory Usage We measure memory usage by providing the Used Tensor Arena Size. The Tensor Arena is a block of memory allocated for use by the model during inference. The used tensor arena size gives the peak runtime memory required for inference. This gives us an indication of the total

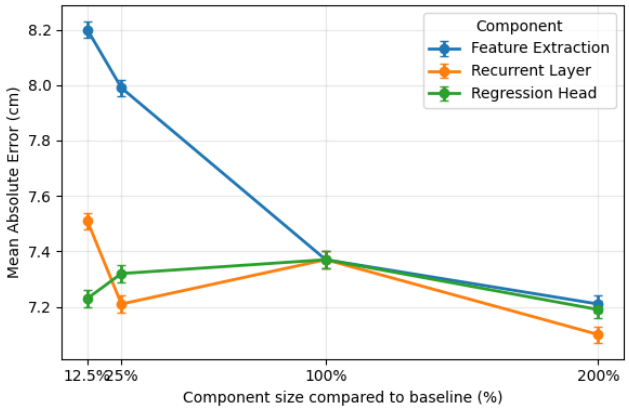

Figure 4: Effect of each component on quantized MAE accuracy

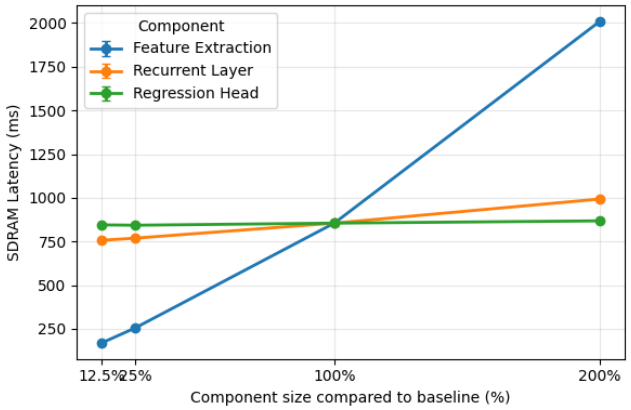

Figure 5: Effect of each component on SDRAM latency

memory required to run the model. This measurement can be found for all quantized models in Table 5.

In Table 4 the size of the unquantized model is shown. This is not an indication of runtime memory, but is the file size of the model.

5 Results

In this section, we will discuss the results of the study through figures. The exact numbers measured in the study, which were used to create these figures, can be found in Appendix A.

Quantized, relative to baseline		
Model name	SRAM Latency	MAE[†]
0125_feat	−68.3%	+11.3%
0250_feat	−56.2%	+8.4%
2000_feat	N/A[‡]	−2.2%
0125_rec	−25.2%	+1.9%
0250_rec	−22.1%	−2.1%
2000_rec	+32.1%	−3.7%
0125_head	−1.4%	−1.8%
0250_head	−2.1%	−0.6%
2000_head	+1.7%	−2.4%
impr_small	−85.9%	+4.8%
impr_medium	−78.6%	+10.3%
impr_large	−22.4%	+1.3%

Table 3: Selected metrics of trained configurations compared to baseline.

[†] ± 0.43%

[‡] the model 2000_feat does not fit in SRAM memory.

5.1 Relative Effects of Each Component on Latency and Accuracy

In figures 4 and 5 we can see the effect of scaling up each component individually on quantized MAE accuracy and SDRAM latency. From these figures, it is clear that the feature extraction component has the greatest influence on both the MAE accuracy and SDRAM latency.

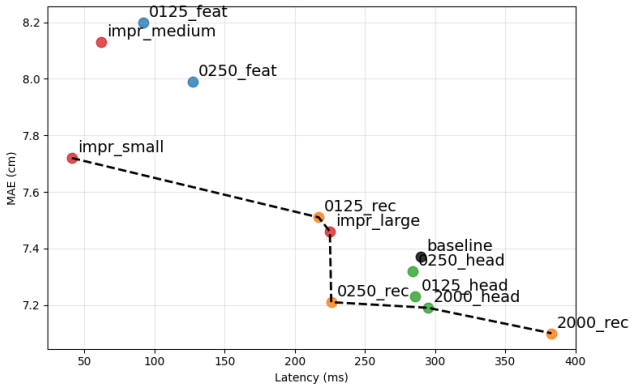

Figure 6: Pareto frontier of all configurations for SRAM latency vs MAE accuracy
excluding 2000_feat, as SRAM latency cannot be measured for this model

Although the feature extraction component is the largest influence identified on model accuracy, from Table 3 it can be observed that this difference is only at most 11.3%. However, the effect on latency is much greater, saving up to 68.3% with the maximum reduction. This could indicate that, while a larger feature extraction component contributes to accuracy, it can be compressed without costing significant accuracy.

From Figure 4 it also becomes clear that the recurrent layer has a fairly unstable effect on accuracy. From our results, no clear relationship between recurrent layer size and accuracy can be drawn. This is an unexpected result, as intuitively we often rely on information from the past to predict the future, which is exactly the task of the recurrent layer. This would lead us to expect the recurrent layer to have a great effect on the accuracy. This discrepancy might indicate that the recurrent layer is wrongly implemented in the baseline model.

Finally, from the figures, it can also be seen that the regression head component has no significant impact on latency, but that reducing it positively impacts accuracy. This might indicate that the regression head is oversized for its task and can therefore not be trained properly.

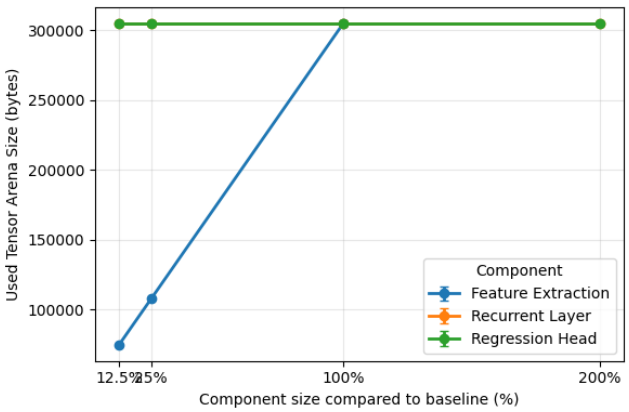

Figure 7: Effect of each component on used tensor arena size

5.2 Pareto Frontier

All trained models, including the proposed models, have been plotted in Figure 6. This figure shows the accuracy and latency of each model and shows a line at the Pareto frontier. The Pareto frontier shows every model that cannot be positively surpassed on all metrics simultaneously.

From this figure it becomes clear that the two likely best results are that of impr_small and 0250_rec. 0250_rec is an overall improvement over the baseline, being both more accurate and offering lower latency.

However, the difference in accuracy of 0250_rec compared to impr_small (6.6%) is relatively small compared to the difference in latency (81.9%). This means that impr_small is likely a better model for most use cases, if losing 4.8% accuracy compared to the baseline model is acceptable.

5.3 Effects of Each Component on Memory Usage

In Figure 7 the used tensor arena size can be seen for each model. The value for 2000_feat was excluded from this figure, as this model did not fit in the arena size.

This figure shows that the feature extraction is the only component that had an influence on the used tensor arena size. This explains why a model with a larger file size, like 2000_rec, was able to fit in the tensor arena size on SRAM, while 2000_head could not. The file sizes of each model can be found in Table 4, in Appendix A.

However, this result is still unexpected. This result could imply that the recurrent layer and regression head are not making good use of the tensor arena. This could be the cause of their lesser effect on model accuracy, though this is only speculative and more research is required to answer these questions.

5.4 Proposed Models

From figures 4 and 5, it is clear that the recurrent layer and regression head are optimal at 25% and 12.5% size compared to the baseline model, respectively. Since the feature extraction has the greatest effect on our metrics, we propose 3 improved models. impr_small uses the smallest form of the feature extraction, focusing on low-latency. impr_large uses the same feature extraction as the baseline model, in order to have less latency than the baseline model while keeping its accuracy. impr_medium will use a feature extraction component of 25% the size of that of the baseline model, in order to find a balance between impr_small and impr_large.

After training these models, we found surprising results. The results for impr_small are around what was expected and is a significant improvement over the baseline model, saving 85.9% of latency at the cost of 4.8% accuracy. However, impr_medium actually costs more accuracy and saves less latency and is therefore made obsolete by impr_small. impr_large turned out to be less accurate and slower than expected and is almost entirely surpassed by 0250_rec.

This makes the proposed models different from what was expected. Instead of proposing impr_small, impr_medium and impr_large, we propose 0250_rec as an overall improvement on the baseline model, and impr_small as a lightweight alternative, saving significant latency at the cost of relatively little accuracy.

6 Responsible Research

As with any research, there are some ethical concerns for this paper. In this chapter, we want to highlight these issues and show why we chose to pursue this research despite these concerns.

6.1 Dataset Integrity and Reproducibility

The dataset used for training can raise various ethical and reproducibility concerns. The foremost issue is that mmWave human-pose estimation datasets are often not open-source. From the eight mmWave human-pose estimation datasets listed by J. Yang et al. [14] we find that only half is open-source and available. This limits our choice of dataset to use, as constructing a dataset specifically for this paper is infeasible with the given budget. In this paper, we choose to use the mRI dataset [1]. mRI is an open-source dataset, which promotes the reproducibility of this paper.

Another concern is the representation of subjects in the dataset. The mRI dataset is represented by 20 subjects, consisting of 13 males and 7 females. This makes for a not-ideal but reasonable gender balance. The average age and height of the subjects is respectively 24.1 ± 4.4 and 175.6 ± 9.3cm, which makes for a subject group that is younger and taller than average. Although this dataset represents most of the population, it still lacks some representation. This is especially a concern for older groups, which are not represented in this group, while they are the main demographic for most of the healthcare applications of mmWave. This should be considered before extrapolating these results to the entire population.

The mRI dataset contains 12 unique actions, consisting of a total of 160k frames. Of these 12 actions, the 11^{th} and 12^{th} respectively are determined by the subject and involve global displacement, making for a diverse dataset. This diversity makes the dataset a more accurate representation of how the situation might look when this technology is deployed outside of research. This contributed to us choosing to use this dataset.

6.2 Model Training Cost

Training deep-learning models often requires a lot of computational resources, leading to high electricity consumption and increased demand for hardware components such as GPUs. This might raise ethical and environmental concerns, especially when training is done excessively. In this paper, these concerns are all addressed: The main focus of this paper is to find a more efficient model than the baseline, which in the long-term likely outweighs the damage done by the trainings. Additionally, we have also made sure to not train excessive amounts of models, such as by opting to use a manual grid-based search instead of optimisation algorithm like Bayesian optimisation.

6.3 Reproducibility of Results

Reproducibility is an important factor in good research. However, training deep-learning models is inherently random, which means that the training dataset and training parameters alone are often not enough to reproduce the exact model. This paper is part of a larger project in collaboration with industry, and it is still being discussed whether the code can be made public. This means the code can not be made open-source at this time. Although the exact model weights might therefore not be reproducible, we believe that retraining the model under similar conditions will produce comparable results in accuracy, latency, and memory usage to our model. Since these are the most important metrics for this study, we consider the proposed methodology to be sufficiently reproducible.

6.4 Sensing through Walls

MmWave radar poses serious privacy risks because its signals can penetrate certain non-metallic materials, such as cardboard, fabric, or thin walls. Because of this, a user may incorrectly assume that covering up the sensor disables its ability to detect movement or presence, when in reality it does not. This may lead to users believing they are not being tracked when they are. This raises concerns of non-consensual tracking and other malicious use. Although we think the contributions of this paper do not significantly increase these concerns, as we are primarily improving a pre-existing model, these privacy concerns should nevertheless be carefully addressed before mass-deployment.

6.5 MmWave Privacy Compared to Cameras

MmWave is often presented as a more privacy-preserving alternative to the camera. Because mmWave-based systems create abstract representations such as skeletal pose rather than capturing visual imagery, they are often assumed to preserve the privacy of the person being monitored. However, this assumption of privacy might be incorrect: Given enough data it could be possible to identify a person through the mmWave radar. This is because the radar might be able to predict not only the skeletal pose, but also body structure, physical mannerisms, and gender [13] of the person, each features that could identify a person. However, it is significantely harder to identify a subject from a mmWave radar compared to an RGB camera, so this is of little concern.

7 Discussion

Our best result is likely that of the impr_small model, with a MEA accuracy of 7.72cm and a latency of 41ms on our embedded device.

This is a significant finding, since the latency competes well with state-of-the-art human-pose estimation models, such as mmMesh with an inference latency of 28ms [13]. While mmMesh is faster, it is GPU based, whereas our model runs on an embedded device. This shows that embedded models can compete with state-of-the-art GPU based models.

However, mmMesh is nearly an order of magnitude more accurate at 2.47cm MPJPE accuracy, compared to 15.77cm for our model. This might be due to constraints from our baseline, such as that the mmMesh dataset is not open source and therefore could not be used in the baseline. This makes for a less fair comparison. It should also be noted that mmMesh is not strictly a human pose estimator but a human mesh estimator.

TinyssimoRadar, an embedded model, achieves a latency of 32.3ms [10], which we consider to be similar to our model. TinyssimoRadar uses mmWave to estimate out of a set of hand gestures which is being performed by the user. It does this with 94.9% accuracy, however since this is a different objective to our model, it is hard to compare these results.

We think the results show significant improvement in latency, even competing with GPU-based models, but the accuracy of the model is still lacking. We consider that more research should be done towards model accuracy before this technology can be deployed.

8 Conclusions and Future Work

This paper discussed the relationship between the sizes of the feature extraction, recurrent layer and regression head components in a mmWave pose estimation model, and its accuracy, latency and memory usage.

We found that the feature extraction component causes the vast majority of latency but also greatly positively impacts accuracy compared to the recurrent layer and regression head. We also found that the regression head contributes very little to latency cost and that using a greater regression head might actually cost accuracy rather than gain it.

Using these results, we found a new model, impr_small, that significantly optimises latency (reducing it by 85.9%) at the cost of 4.8% accuracy. In general, we found the model size can be greatly reduced without costing significant accuracy.

We also proposed another model that we found, 0250_rec, as an overall improvement on the baseline model, as it offers both lower latency and more accuracy at the same memory usage.

These results raise further questions, such as about the contribution of the recurrent layer to accuracy. A larger recurrent layer occasionally costs accuracy, while gaining accuracy other times (e.g. model 0250_rec is more accurate than the baseline, however, it has a smaller recurrent layer). This is surprising, as intuitively we often rely on information from the past to predict the future, which is exactly the task of the recurrent layer. This might indicate that the recurrent layer is

wrongly implemented in the baseline model. This is something that can be addressed in further research.

Another question might be that of the relation between the layers inside each component. In this paper, we only looked at scaling entire components, but each component is made up of several layers which can be scaled individually. Not only that, the number of layers can also be altered, adding even more potential for optimisation.

Acknowledgements

We would like to thank Marco Zúñiga Zamalloa and Nicole Rosi for their guidance and supervision for this thesis. We also thank DelftBlue for the computational resources and Nicole Rosi for providing the embedded device.

References

[1] Sizhe An, Yin Li, and Umit Ogras. mri: Multi-modal 3d human pose estimation dataset using mmwave, rgb-d, and inertial sensors, 2022.

[2] Sizhe An and Umit Y. Ogras. Mars: mmwave-based assistive rehabilitation system for smart healthcare. *ACM Trans. Embed. Comput. Syst.*, 20(5s), 2021.

[3] James Bergstra, Remi Bardenet, and Balázs Kégl. Algorithms for hyper-parameter optimization. 2011.

[4] Anjun Chen, Xiangyu Wang, Shaohao Zhu, Yanxu Li, Jiming Chen, and Qi Ye. mmbody benchmark: 3d body reconstruction dataset and analysis for millimeter wave radar. In *Proceedings of the 30th ACM International Conference on Multimedia*, MM '22, page 3501–3510. Association for Computing Machinery, 2022.

[5] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale, 2018.

[6] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, 2014.

[7] Feng Jin, Arindam Sengupta, and Siyang Cao. mmFall: Fall Detection Using 4-D mmWave Radar and a Hybrid Variational RNN AutoEncoder. *IEEE Transactions on Automation Science and Engineering*, 19, 2022.

[8] Shih-Po Lee, Niraj Prakash Kini, Wen-Hsiao Peng, Ching-Wen Ma, and Jenq-Neng Hwang. Hupr: A benchmark for human pose estimation using millimeter wave radar. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5715–5724, 2023.

[9] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[10] Andrea Ronco, Philipp Schilk, and Michele Magno. TinyssimoRadar: In-Ear Hand Gesture Recognition with Ultra-Low Power mmWave Radars. pages 192–202. IEEE, 2024.

[11] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[12] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, 2011.

[13] Hongfei Xue, Yan Ju, Chenglin Miao, Yijiang Wang, Shiyang Wang, Aidong Zhang, and Lu Su. mmMesh: towards 3D real-time dynamic human mesh construction using millimeter-wave. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 269–282, 2021.

[14] Jianfei Yang, He Huang, Yunjiao Zhou, Xinyan Chen, Yuecong Xu, Shenghai Yuan, Han Zou, Chris Xiaoxuan Lu, and Lihua Xie. Mm-fi: Multi-modal non-intrusive 4d human dataset for versatile wireless sensing, 2023.

A Numerical Results in Tables

After training each model, the metrics were measured, both for the quantized and unquantized versions of the model. The measurements for unquantized and quantized accuracy can be found, respectively, in tables 4 and 5. The measurements for quantized latency can be found in Table 6. Model file size and memory usage can be found, respectively, in tables 4 and 5.

This appendix contains the raw numbers from the study. These results are made clearer and are interpreted in section 5.

	Unquantized		
Model name	MAE† (cm)	MPJPE† (cm)	Size (Bytes)
baseline	6.97	14.14	239,856
0125_feat	8.01	16.47	162,024
0250_feat	7.62	15.60	170,216
2000_feat	6.86	13.84	364,616
0125_rec	7.41	14.97	89,000
0250_rec	7.13	14.25	104,688
2000_rec	6.69	13.55	536,136
0125_head	6.94	14.17	193,712
0250_head	7.04	13.99	200,304
2000_head	7.02	14.01	292,592
impr_small	7.67	15.66	44,072
impr_medium	7.40	15.02	46,248
impr_large	7.23	14.69	78,896

Table 4: Accuracy and size metrics of trained configurations before quantization.

† $\pm$ 0.03 cm

	Quantized		
Model name	MAE† (cm)	MPJPE† (cm)	Arena (Bytes)
baseline	7.37	15.46	305,124
0125_feat	8.20	17.41	74,756
0250_feat	7.99	17.32	107,652
2000_feat	7.21	15.59	N/A‡
0125_rec	7.51	15.31	305,124
0250_rec	7.21	14.71	305,124
2000_rec	7.10	15.67	305,124
0125_head	7.23	15.88	305,124
0250_head	7.32	15.60	305,124
2000_head	7.19	14.67	305,124
impr_small	7.72	15.77	74,756
impr_medium	8.13	18.99	107,652
impr_large	7.46	16.50	305,124

Table 5: Accuracy and size metrics of trained configurations after quantization.

† $\pm$ 0.03 cm

‡ the model 2000_feat does not fit in SRAM memory.

	Quantized	
Model name	SRAM Latency (ms)	SDRAM Latency (ms)
baseline	290	855
0125_feat	92	170
0250_feat	127	255
2000_feat	N/A†	2010
0125_rec	217	757
0250_rec	226	769
2000_rec	383	993
0125_head	286	845
0250_head	284	843
2000_head	295	868
impr_small	41	97
impr_medium	62	172
impr_large	225	762

Table 6: Latency metrics of trained configurations after quantization.

† the model 2000_feat does not fit in SRAM memory.