LTNIN: Lightweight Transformer Network for Inertial Navigation

Xin Li, Jiayu Zhou*, Peng Ye, Kai Dong
School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, China
Emails: linuxcumt@126.com, zhoujy@cumt.edu.cn, yepcumt@126.com, dk6857@sina.com
** Corresponding author*

Abstract: **Pedestrian dead reckoning is an important task for smartphones and smart wearable devices. Traditional gait model-based methods make it difficult to match multiple motion patterns of different users. The performance of inertial odometry based on deep learning is impressive.At present, most of the data-driven pedestrian dead reckoning backbone networks use Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTMs), Residual Network(ResNet) and Temporal Convolutional Network(TCN) and related variants. In recent years, Transformer architecture has greatly progressed in large models and multi-scenes. However, the Transformer has many parameters and is not friendly to mobile edge devices. Given the demand for edge devices for lightweight models and the time series characteristics of inertial sensor data, this paper designs a lightweight Transformer inertial odometry network based on LightViT for edge devices. It uses group convolution to extract the characteristics of IMU data while reducing network parameters and fuses IMU data time and space information through a multi-head self-attention mechanism and bidirectional attention multi-layer perceptron. It is verified on three mainstream data sets (RONIN, OXIOD, RIDI). Compared with the ResNet network, which has the best effect in RONIN, ATE is reduced by 8% on average. When the network parameters are reduced by 76.6%, FLOPs are reduced by 77%, and the inference time is 12% faster. With comparable accuracy to ResNet, the network reduces parameters by 87%, FLOPs by 86%, and accelerates inference by 76%. It provides a new reference for future backbone network selection in resource-constrained pedestrian dead reckoning systems.**

Keywords: **Pedestrian Dead Reckoning(PDR), Deep Learning,Light Transformer, Inertial Navigation.**

I. INTRODUCTION

In terms of smart phones and smart wearable devices, rapid and accurate pedestrian trajectory estimation is required under many complex conditions, including indoor and outdoor daily trajectory backtracking, tourism reality explanation, emergency rescue, underground parking lot location sharing, human-computer interaction and augmented reality. Pedestrian position changes are flexible and random. For example, pedestrians walking in the city may pass through dense streets, buildings, and subway channels in a short period of time. The scene is not limited to indoor or outdoor. The frequent transformation of GNSS signals leads to weak or even no signals, which is not conducive to continuous trajectory recording in complex environments. Inertial sensors are widely used in anomaly detection tasks due to their independence and weak dependence on the environment, such as fall detection [1], activity recognition [2], and are also regarded as an indispensable part of pedestrian dead reckoning in complex scenes [3].

Nowadays, almost every mobile device is equipped with cheap MEMS sensors. By collecting the angular velocity measured by the gyroscope in the inertial measurement unit (IMU) and the linear acceleration measured by the accelerometer, the inertial navigation system (INS) based on Newton 's second law can directly calculate the trajectory of the object. However, low-cost inertial sensors have strong noise, and the position cumulative error of INS will diverge quadratically with time, so it is difficult to achieve long-term effective position estimation on MEMS sensors.

The PDR algorithm uses inertial sensor data to estimate the user's step frequency, step size, and direction of motion. Class algorithms are often complicated to adapt to the walking habits of each user. The detection of the step frequency is realized by the periodicity of the acceleration signal, which limits the use of the PDR [4] algorithm.

The emergence of deep learning provides new possibilities for extracting information from IMU data. Recent studies have shown that data-driven inertial odometers can provide trajectories by returning average speed to sensor data. As an innovative inertial navigation method, RIDI [5] first proposed the use of machine learning technology to improve the accuracy of position estimation using inertial measurement units (IMUs). The model focuses on the robust integration of linear acceleration and angular velocity data to predict the trajectory and uses linear least squares to refine IMU data to correct differences. IONet [6] pioneered the transformation of the challenge of inertial navigation into a continuous time series learning paradigm. By returning speed and direction directly from the network output, the traditional integration method is avoided. Its architecture promotes an innovative way of processing IMU data to ensure effective prediction of trajectory without relying on conventional sensor fusion technology. RoNIN [7] has established a solid framework for neuro-inertial navigation, and the introduction of RoNIN variants based on ResNet [8], LSTM [9], and TCN [10] significantly improves motion estimation under natural human activities. Recently, Transformer [11] has been widely used in computer vision (CV) [12-14], NLP [15], long-time series prediction [16,17], and large language model [18], which shows its powerful ability to model the long-term dependencies between input sequence elements. In addition, the Transformer model can also combine visual, audio and other modal data to improve the recognition accuracy of

trajectory estimation. However, it needs to store and calculate a large number of attention matrices when dealing with long sequences, resulting in significant memory consumption [19], making it not suitable for dealing with inertial positioning tasks on mobile terminals. However, with the rise of the ChatGPT series of large models, more and more mobile phone manufacturers have begun to study the use of large models to reconstruct the underlying operating system. At present, common large models are based on the Transformer architecture. CTIN [20] is the first inertial navigation model to introduce Transformer architecture. By combining the ResNet encoder and Transformer decoder, a multi-head attention mechanism is used to fuse spatial and temporal information, to predict speed and trajectory more accurately. Due to the large amount of calculation of the Transformer architecture, it is not friendly to the mobile device code. Recently, lightweight Transformers [21-24] have achieved good results in processing mobile vision tasks, showing the potential to replace CNNs and traditional Transformer architectures on mobile devices. These architectures can bring a better balance between performance and accuracy, and a network that takes into account higher accuracy and lower computing power has always been the pursuit of researchers.At the same time, the lightweight Transformer architecture also shows its adaptability in the field of Human Activity Recognition(HAR) [25], reflecting its potential for processing inertial sensor data.

In addition, the pedestrian dead reckoning task can also perform some deeper data mining by recording the trajectory, such as inferring the user 's living habits, consumption habits, and even mental state by changing the trajectory over a period of time. The Transformer architecture has shown certain advantages in these fields. The introduction of the Transformer architecture in the pedestrian dead reckoning task in advance can reduce some repetitive data information mining work and provide the possibility for subsequent multi-task processing. At the same time, existing research shows that relying solely on deep neural networks for pedestrian dead reckoning may face problems such as difficulty in returning to 3D displacement[26]. It is also necessary to combine traditional algorithms such as EKF, or combine multi-sensors such as magnetometers for data fusion[27,28]. This paper intends to improve the effect and efficiency of the backbone network of the data-driven pedestrian dead reckoning system, and provide a new reference for the backbone network upgrade of the previously mature pedestrian dead reckoning system and the backbone network selection of later methods. In order to eliminate the interference of other factors, the system does not use mature traditional methods to supplement the overall system, but also improves the versatility and portability of the system.

In this paper, we propose a lightweight inertial odometer for smart mobile devices. The main contributions of this paper are summarized as follows :

1. Based on the LightViT network, an inertial odometer network LTNIN-Net based on lightweight Transformers is proposed. The compatibility of lightweight Transformers for pedestrian dead reckoning tasks is proved, which provides a new reference for selecting a backbone network for similar tasks.

2. When the ATE is reduced by 8 % compared with the RONIN-optimal ResNet network on three public datasets, the FLOPs are reduced by 77 %; when the accuracy is the same, FLOPs are reduced by 86 %.

The rest of this article is structured as follows. Section II describes the whole system in detail. Section III introduces our experimental environment settings and network learning configuration in detail. Section IV introduces the evaluation results of the performance, efficiency, and robustness of the studied model on three related public datasets. Section V summarizes the whole paper and puts forward the prospect of future work. We represent the proposed system as LTNIN and the lightweight Transformer-based inertial odometry network as LTNIN-Net for further description.

II. METHOD

We designed a network to estimate the speed based on IMU data input, and then integrate the speed to generate the position. This network architecture contains multiple modules, in which the input data is extracted by the convolutional embedding module (ConvStem), and the deep feature learning is performed by multiple hybrid converter blocks (HybridAttnBlock) to effectively capture the timing information in the speed estimation. The velocity estimation result is used to calculate the position of the object through the integral process. In order to ensure the reproducibility of the method, this section describes in detail the network architecture and the design ideas of each module, hyperparameter selection, training process and experimental settings. The design decisions are elaborated, and the necessary code and configuration files are provided to ensure that other researchers can reproduce our experimental results.

A. Speed assessment network

1) The overall structure of the model:

We propose a neural network to learn the relationship between the original IMU data and the horizontal velocity vector. The network structure is simplified and modified based on LightViT. While retaining its characteristics of dealing with multivariate time series regression, the network structure is simplified to reduce the number of parameters, and the unique part of the processed image is changed to deal with one-dimensional data. The whole network structure is shown in Fig.1.

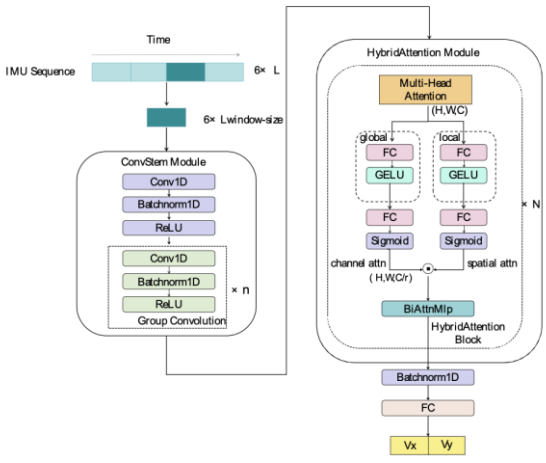

Fig. 1. Overview of LTNIN-Net architecture

The input of the system is the original IMU measurement, that is, $L\times$ 6-dimensional linear acceleration (a) and angular rate (ω_n) sequence. The heading-independent coordinate system (i.e., any coordinate system aligned with the axis and gravity) is used to represent the input original IMU data and the output velocity data. $L_{window-size}$ is the size of the sample window, the setting of $L_{window-size}$ here is consistent with the previous method and is set to 200. The output of the network is a two-dimensional velocity vector : velocity v_x in the horizontal X-axis direction and velocity v_y in the horizontal y -axis direction. We learn the relationship between the original IMU measurement and the velocity increment vector through the deep neural network. In addition to the easy modeling, These relationships include not only $v_0 = at$ which is easy to model, but also other IMU noises which are not easy to model.

LTNIN-Net is mainly composed of two parts. The first part uses multiple convolutional and grouped convolutional layer sequences to extract the features of the input sequence, which can reduce the network FLOPs while retaining the key dynamic information in the IMU data. This design is critical to reduce the early computing requirements of the network and does not significantly lose information. The second part uses a series of hybrid converter blocks (HybridAttnBlock). Each block contains an Attention module and a Bidirectional Attention Multilayer Perceptron (BiAttnMlp) module, which aims to process local and global features by combining self-attention mechanism and Multilayer Perceptron (MLP). The self-attention mechanism can capture long-distance dependencies and synthesize them with local features to solve the noise and error problems in IMU data. This is particularly important for the pedestrian dead reckoning task, because the pedestrian's movement pattern has strong temporality and local dependence.

We use the mean square error (MSE) as the loss function in the training process. The MSE loss is defined as :

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\left|\hat{y}_i^{(Vx)} - y_i^{(Vx)}\right|^2 + \left|\hat{y}_i^{(Vy)} - y_i^{(Vy)}\right|^2\right) \quad (1)$$

where $\hat{y}_i^{(Vx)}$ and $\hat{y}_i^{(Vy)}$ are the speeds of the x-axis and y-axis of the output of the network at time i. $y_i^{(Vx)}$ and $y_i^{(Vy)}$ are the corresponding ground real velocities, which are obtained from other sensors such as GNSS or cameras. N is the total number of data in each round of the training set.

2）Feature extraction convolution (ConvStem) module:

The ConvStem module first performs preliminary feature extraction and dimensionality reduction on the sequence data through a layer of convolutional network. The module first receives a data with a fixed sequence length $L_{window-size}$ and a specified number of input channels (for example, 6). These data are from IMU sensors and contain information such as acceleration and angular velocity. In the first layer of the module, an ordinary one-dimensional convolutional layer is used to extend the input channel to a higher dimension, aiming to provide sufficient feature representation capabilities for subsequent deep learning modules. The convolution operation of this layer can be expressed as :

$$X_{out} = \text{Conv1D}(x, W, b) \quad (2)$$

Among them, W and b are convolution kernel and bias term respectively, Conv1D represents one-dimensional convolution operation, and xis input data. The convolution operation generates a high-dimensional feature representation by sliding convolution of the input data with the convolution kernel. Then, the convolution output is standardized by batch normalization (BatchNorm1D) to ensure the stability of the training process and prevent the gradient from disappearing or exploding. Subsequently, a nonlinear transformation is introduced by the activation function to enhance the expression ability of the model.

After the initial one-dimensional convolution, we designed six grouping convolution layers. Each group convolution layer divides the input features into six groups, and performs independent convolution operations within each group. This grouping convolution helps to reduce the number of parameters of the model [19], thereby reducing the risk of overfitting and improving computational efficiency. After each convolution operation, a batch normalization layer and a ReLU activation function are followed. The batch normalization layer is used to standardize the processing features to prevent the gradient from disappearing or exploding during the training process, while the ReLU activation function introduces nonlinearity and enhances the model 's ability to express data. The convolution operation of each group is :

$$X_{grouped} = \text{GroupedConv}(x_{conv}, W_{group}, b_{group}) \quad (3)$$

Among them, W_{group} and b_{group} represent the convolution kernel and bias term of each group of convolutions respectively, GroupedConv represents the grouping convolution operation, andx_{conv} is the feature after preliminary convolution and batch normalization.

Through this setting, each layer can effectively transform and enhance feature expression while keeping the length of the sequence unchanged. After multiple convolution, normalization and activation layers, the high-dimensional features are remapped back to the lower dimension through a 1D projection convolution layer, so that the dimension of the output features matches the input requirements of the subsequent modules of the network. For example, if the input is divided into 6 groups, the convolution operation of each group can be performed independently, which can reduce the number of parameters and improve efficiency. Suppose that the number of channels in each group is $\frac{C_{in}}{6}$, then the output Y of the group convolution can be expressed as :

$$Y = \text{ReLU}\left(\text{BN}\left(W_g * X_g + b_g\right)\right) \quad (4)$$

Among them, W_g, b_g and X_g represent the convolution kernel,bias,and input segments of the group, respectively.

Then, the feature dimension is adjusted from the output dimension C_{out} of Y to the expected output dimension to C_{emb} through a 1D convolutional projection layer :

$$Z = W_p * Y \quad (5)$$

where W_p is the projection convolution kernel, Z is the output after projection, and has the expected output dimension C_{emb}. The projection layer maps the dimension of Y from C_{out} to the dimension C_{emb} of Z through W_p.

Finally, the projected features are normalized by an a LayerNorm layer to ensure the stability of the output and eliminate the scale difference between different features. This

process helps to improve the training efficiency of the model and accelerate the convergence. The output of the module is sent to the subsequent HybridAttnBlock module for deeper feature learning.

3）HybridAttnBlock module:

HybridAttnBlock integrates the traditional multi-head self-attention (Attention) and Bidirectional Attention Multilayer Perceptron (BiAttnMlp) mechanisms. This module strengthens the processing of input features and improves the model 's ability to capture local and global feature information. It is especially suitable for tasks such as IMU data that need to deal with complex time series data and long-distance dependence. The Attention module allows the model to capture long-distance dependencies in the input data, which is one of the core advantages of the Transformer model. The features of channel and spatial dimensions are not only very important in the field of image, but also proved to be important in long-term feature extraction. The features of channel and spatial dimensions are not only very important in the field of image, but also proved to be important in long-term feature extraction. The dual-head attention in the BiAttnMlp module dynamically focuses on the important features of cross-channel and spatial dimensions, and enhances the expression ability of Feed-Forward Network (FFN) through global reduction and local reduction techniques.At the same time, we add a linear reduction layer before the fully connected layer of the attention mechanism to perform dimension reduction. The reduction rate r is 4, so that the number of output channels is one-quarter of the number of input channels, thereby reducing the FLOPs of subsequent operations.

The Attention module is the core of the Transformer architecture, which allows the model to capture information at any position within the sequence. This core part uses dot product attention to calculate the internal dependency of the input sequence. First, input features X are linearly transformed into queries (Q), keys (K), and values (V) :

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \quad (5)$$

Among them, W^Q, W^K, W^V is the weight matrix, corresponding to the query, key,and value respectively, Q, K, V is obtained by linear transformation of input X respectively. Then the dot product score between the query and the key is calculated and normalized by the scaling factor $\sqrt{d_k}$.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (6)$$

Here, d_k is the dimension of the key vector, and the softmax function is used to convert the attention score into a probability distribution. Finally, the above scores are used to weight the value matrix to generate output features. This module helps the model capture long-distance information,and each 'head' focuses on different subspaces of the input. At the same time, Dropout is also applied to the attention weight to enhance the generalization ability of the model and prevent overfitting.

The BiAttnMlp module is another key part of the HybridAttnBlock, which enhances the expression ability of the FFN by combining local (channel) and global (spatial) attention. The design of BiAttnMlp enables the model to fine-tune the attention to different features at each layer. The following is the specific implementation :

Channel attention is calculated by global features in order to dynamically focus on the important channels in the input features. Specifically,the input X is transformed through two linear layers and the GELU activation function is transformed. One layer is used for channel attention, and the other layer is used for spatial attention features to enhance the nonlinear expression ability of the network :

$$X_{\text{global}} = \text{GELU}\left(W_{\text{global}}X + b_{\text{global}}\right) (7)$$

$$X_{\text{local}} = \text{GELU}(W_{\text{local}}X + b_{\text{local}}) \quad (8)$$

The channel attention of the global feature is calculated, and the sigmoid function is used to obtain the weighting coefficient.

$$C_{\text{attn}} = \sigma\left(W_c\left(X_{\text{global}}\right)\right) (9)$$

The calculation method of spatial attention is similar to channel attention, but it generates weighting coefficients by stitching global and local features :

$$S_{\text{attn}} = \sigma\left(W_s\left(\left[X_{\text{local}}, X_{\text{global}}\right]\right)\right) (10)$$

Finally, channel and spatial attention are applied to input features respectively, and these weighted features are integrated by element multiplication :

$$X_{\text{final}} = (C_{\text{attn}} \cdot S_{\text{attn}}) \cdot X \quad (11)$$

This enables the model to strengthen important features, thereby improving the accuracy and richness of feature expression.

The multi-head self-attention in HybridAttnBlock enables the model to process different parts of the sequence in parallel and capture a wide range of dependencies. The bidirectional attention further refines the model's attention to specific regions (such as specific channels or spatial locations), making the feature expression more abundant and accurate. Finally,superimposing N HybridAttnBlocks,not only enhances the expression ability of the model but also improves its ability to understand complex data patterns.

B. Position estimation

Position estimation involves starting from the initial known position and calculating its position in space by integrating the speed of the moving subject at each moment. Specifically, the LTNIN network described in the previous section can return the speed of each moment. We can calculate the moving distance and direction of the object by integrating these speed data to determine its final position. The mathematical description of the integral method is as follows: Assuming that the initial position of the object is $\left(p_{x_0}, p_{y_0}\right)$, and the velocity vector at time t is $v(t) = \left(v_x(t), v_y(t)\right)$, then the position $p(t) = \left(p_x(t), p_y(t)\right)$ of the object at time t can be calculated by the following integral formula:

$$p_x(t) = p_{x_0} + \int_0^t v_x(t)dt \quad (12)$$

$$p_y(t) = p_{y_0} + \int_0^t v_y(t)dt \quad (13)$$

Because we are dealing with discrete velocity data points. If the measured values of velocity ti at each discrete time point are v_{x_i} and v_{y_i}, respectively, and the time interval between each time point is Δt, the discrete calculation formula of position is :

$$p_{x_i} = p_{x_0} + \sum_{k=1}^{i} v_{x_k} \Delta t \ (14)$$

$$p_{y_i} = p_{y_0} + \sum_{k=1}^{i} v_{y_k} \Delta t \quad (15)$$

This calculation method is suitable for data collected from sensors such as IMU (Inertial Measurement Unit) and is common in mobile devices and vehicle navigation systems.

III. EXPERIMENTS AND RESULTS

In this section, we compare three versions of LTNIN-Net, denoted as LightNiT-S, LightNiT-R,and LightNiT. All methods are compared on the test set, and these data are not used in the training phase. In addition to the network definition and the setting of training hyperparameters, all methods use a consistent configuration. This section is organized as follows. IV-A introduces the data set used. Section IV-B describes the indicators for evaluating trajectory accuracy. Section IV-C describes the training settings of the experiment. Section IV-D compares the accuracy of existing methods. Ablation experiments were performed in section IV-E.

A. Dataset

In order to be consistent with previous studies, we also evaluate our network on the three most common datasets, namely RIDI, OXIOD,and RoNIN datasets. The detailed description is as follows:

RIDI(Robust IMU Double Integration) data set aims to improve inertial navigation by robustly integrating IMU data. The data set includes IMU sensor measurements and 3D motion trajectories of various human subjects and various device placement methods (such as hand-held and pocket). RIDI uses visual inertial track measurement to collect real ground motion data using a smartphone equipped with Google Tango technology, focusing on various motions including lateral motion, backward and forward walking, and acceleration/deceleration. These diverse motion types ensure that our network is thoroughly evaluated under different dynamic gait phases rather than a single uniform walking pattern.

OXIOD（Oxford Inertial Odometry Dataset）dataset [29] is used for inertial navigation tasks. Data acquisition settings involve a variety of mobile phone placement methods, such as handheld, bag, pocket, and trolley. The high-precision Vicon motion capture system is used to capture IMU sensor data and motion trajectories to provide accurate ground real data. It involves a total of 14.7 hours of data from five human subjects, specially designed to test inertial track measurement algorithms under different mobile phone placement methods and motion conditions. We selected six mobile phone placements to evaluate our work. The inclusion of multiple distinct human subjects and heterogeneous phone carrying modes (such as bags and trolleys) allows us to validate the cross-subject generalizability and behavioral stability of our model under varying degrees of motion constraints.

RoNIN（Robust Neural Inertial Navigation）dataset is a recent open-source data set in the field of inertial navigation research, which surpasses other data sets in terms of scale, diversity, and fidelity. It includes 42.7 hours of IMU motion data from 100 human subjects in different natural mobile phone processing scenarios (such as in a bag, deep in a pocket, or handheld). The data set uses a dual-device data acquisition protocol. One mobile phone is fixed on the body for accurate 3D tracking, while the other mobile phone operates freely for IMU data collection. This setting helps to simulate closer to the real scene. Since the dataset only discloses 50 % of the data, we use 50 % of the data to evaluate our work. Crucially, featuring 100 different human subjects, RoNIN provide unprecedented user diversity and behavioral variations. Testing on this large-scale dataset implicitly confirms that our proposed LTNIN is highly robust against gait variations and individual physiological differences among a vast user cohort.

B. Evaluation Metric

In the inertial measurement unit (IMU) and visual inertial odometer system, the two key indicators for evaluating the accuracy of trajectory estimation are absolute trajectory error (ATE) and relative trajectory error (RTE). This paper uses these two indicators to evaluate the effect of the model.

Absolute trajectory error (ATE) measures the global consistency between the estimated trajectory and the real trajectory. It reflects the overall deviation between the predicted position and the real position in the entire trajectory.

$$\mathrm{ATE} = \sqrt{\frac{1}{n}\sum_{k=1}^{n} \|p_t - \hat{p}_t\|^2} \quad (16)$$

Among them, p_t is the real location of the kth sampling point. $\hat{p}_t$ is the corresponding estimated position. n is the total number of sample points. This formula quantifies the error through the root mean square error, which can effectively reflect the overall difference between the estimated trajectory and the real trajectory.

Relative trajectory error (RTE) is used to measure the error of position change within a fixed time interval or space interval. This reflects the accuracy of the system's dynamic change processing and is not affected by the global position offset. In our experiment, Δt is set to 1 minute.

$$\mathrm{RTE} = \sqrt{\frac{1}{n-\Delta t}\sum_{k=1}^{n-\Delta t} \|(p_{t+\Delta t} - p_t) - (\hat{p}_{t+\Delta t} - \hat{p}_t)\|^2} \quad (17)$$

C. Training Configuration

All models were implemented in the PyTorch framework and trained using the Adam optimizer on a 4 GB NVIDIA GeForce GTX 1650 GPU and a single-core Intel Core i7-9750H CPU @ 2.60 GHz.We use the initial learning rate of 0.0001 for training, and the batch size is 128. We adopt a performance-based learning rate adjustment strategy, that is, when the loss on the validation set does not improve in 10 consecutive cycles, the learning rate is multiplied by 0.1. This method of dynamically adjusting the learning rate allows us to flexibly adjust the learning progress according to the performance of the model on the validation set to obtain better training results. At the same time, we use the dropout layer to effectively avoid network overfitting. All our model training uses this strategy.

D.System Performance

TABLE I. POSITION EVALUATION

	Test subjects	Metric	PDR	IONet	RONIN			LightNiT
					ResNet	LSTM	TCN	
RIDI Dataset	Seen	ATE	3.62	11.61	1.65	1.69	1.58	**1.46**
		RTE	4.71	14.33	1.93	2.01	2.13	**1.77**
RONIN Dataset	Seen	ATE	25.12	29.76	3.86	4.94	4.67	**3.68**
		RTE	21.76	23.51	2.74	2.78	2.71	**2.61**
	Unseen	ATE	22.86	25.07	5.98	6.79	6.33	**5.26**
		RTE	23.15	22.33	4.55	4.55	4.59	**4.41**
OXOID Dataset	Seen	ATE	6.82	1.73	1.66	2.66	4.35	**1.6**
		RTE	3.23	1.57	1.31	2.54	2.98	**1.3**
Our Improvement	Seen	ATE	59.7%	87.4%	11.5%	13.6%	7.6%	RIDI Dataset
		RTE	62.4%	87.6%	8.3%	11.9%	16.9%	
	Seen	ATE	85.4%	87.6%	4.7%	25.5%	21.2%	RONIN Dataset
		RTE	88.0%	88.9%	4.7%	6.1%	3.7%	
	Unseen	ATE	77.0%	79.0%	12.0%	22.5%	16.9%	
		RTE	81.0%	80.3%	3.1%	3.1%	3.9%	
	Seen	ATE	76.5%	7.5%	3.6%	39.8%	63.2%	OXOID Dataset
		RTE	59.8%	17.2%	0.8%	48.8%	56.4%	
	Average	ATE	73.3%	65.4%	8.0%	25.4%	27.2%	ALL Dataset
		RTE	291.1%	68.5%	4.2%	17.5%	20.2%	

We apply the method to the data set introduced in Section III-A. Among them, RIDI and OXOID provide a test set, and a RONIN data set We provide two test sets, one for topics that are also included in the training set, and the other for invisible topics. We compare the proposed model with the three variants of the mainstream methods: IONet and RoNIN on the above data sets. All models are trained to fully converge.

Table I shows our main results. We compare five inertial positioning methods: three variants of PDR, IONet, and RoNIN, and the differences between our methods on three public datasets: the RIDI dataset, OXOID dataset, and RoNIN dataset. We roughed up the best results.The results from the table show that our method achieves the best ATE and RTE on most datasets. Compared with RoNIN-ResNet, RoNIN-LSTM, and RoNIN-TCN, our method improves ATE by 8 %, 25.4 %, and 27.2 % on average, and RTE by 4.2 %, 17.5 %, and 20.2 % on average. Our method has the shortest training time in each data set. Figure 2 shows the visualization of the reconstructed trajectory and the ground truth value, which proves the universal applicability of our method.

TABLE II. COMPARISON OF COMPUTATION EFFICIENCY

Model	ResNet	LSTM	TCN	LightNiT-S	LightNiT-R	LightNiT
Parameters(M)	4.634	0.206	0.371	0.355	0.598	1.086
FLOPs(M)	38.25	41.64	141.21	3.82	5.47	8.78
GPU Inference Time (ms)	5.5	6.1	4.3	2.01	3.13	4.82
Single-core CPU Inference Time (m	300.22	289.17	423.16	170.4	183.19	211.12

Table II shows the comparison of computational efficiency. The comparison of FLOPs, network parameters and inference time of each model is provided. FLOPs represent the number of floating-point operations required for model execution. On mobile devices, models with high FLOPs may require more processing time, because the CPU or GPU of the mobile device is usually weaker than the processing power of the desktop or server. We use GPU (NVIDIA GeForce GTX 1650, 4GB memory) and single-core CPU (Intel (R) Core (TM) i7-9750H CPU @ 2.60GHz) to calculate the inference time 1000 times per model inference.

As can be seen from Table II, the parameter quantity of our best-performing LightNiT model is significantly lower than that of ResNet in RONIN, while its FLOPs are substantially reduced by 77% and 79% compared with ResNet and LSTM, respectively. To better accommodate edge deployments with strict hardware constraints, we further introduce smaller variants, namely LightNiT-S and LightNiT-R. As noted, these smaller variants inevitably incur a slight trade-off in tracking accuracy compared to the full LightNiT or ResNet. However, they yield massive improvements in efficiency: the FLOPs of LightNiT-S are drastically reduced by 90% and 91% compared with ResNet and LSTM, respectively. Although hardware parallel efficiency varies between GPUs and CPUs, the overall computational overhead is substantially reduced.This shows that the LightNiT series model is more suitable for scenarios that require fast processing or resource-constrained devices.

In terms of single-core CPU inference time, which serves as a highly conservative baseline to simulate resource-constrained mobile processors, LightNiT-S and LightNiT required only 170.4 ms and 211.12 ms per window inference, respectively. This represents a significant latency reduction of up to 43.2% compared to conventional architectures like TCN (423.16 ms). Crucially, considering that our framework processes inertial data within a sliding window framework that updates at a standard interval (e.g., 1000 ms), a single-core CPU execution time of ~200 ms guarantees that the model utilizes only a fraction (approx. 20%) of a mobile processor's single-core capacity to keep pace with real-time data streaming. This quantitatively demonstrates that the LightNiT series is exceptionally well-suited for non-blocking, real-time deployment on mobile terminals, thoroughly mitigating the resource-heavy penalty of conventional transformers.

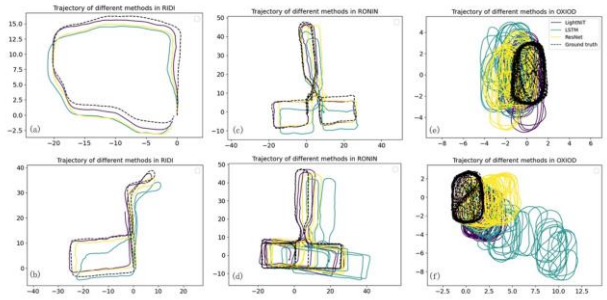

Fig.2. Visualization trajectories of different methods, in which the real trajectory is marked as a black dotted line, and LightNiT, RONIN-ResNet, and RONIN-LSTM are marked as purple, yellow, and dark green, respectively.

Figure 2 shows the inference trajectory of LigthNiT, RONIN-LSTM, RONIN-ResNet and the visualization of the real trajectory. We select two examples from each dataset and visualize their trajectories. Where (a) and (b) are from the RIDI dataset. (c) and (d) are from the visible and invisible sets of the RONIN dataset, respectively. (e) and (f) are from the OXOID dataset.

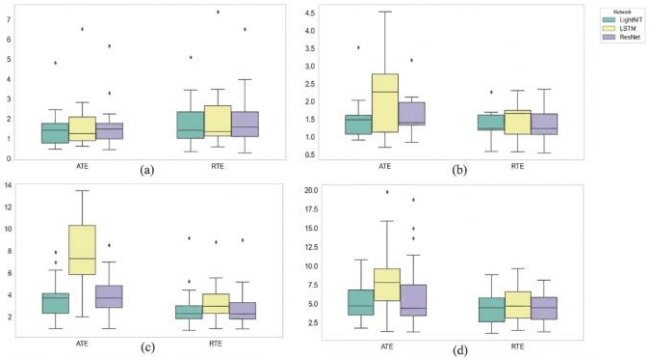

Fig.3. ATE and RTE index box plots of different networks in the selected test set are represented by dark green, yellow and purple, respectively, LightNiT, RONIN-LSTM and RONIN-ResNet networks. The rhombus black block represents the outliers.

Figure 3 shows the block diagram of ATE and RTE in the test set of LightNiT, RONIN-LSTM and RONIN-ResNet networks. In these four data sets, our method is superior to the comparison method. The data distribution is more concentrated than other methods, with the smallest ATE and RTE, less data fluctuation, and the least outliers. The results of the dataset show that our method is stable, reliable and high-performance.

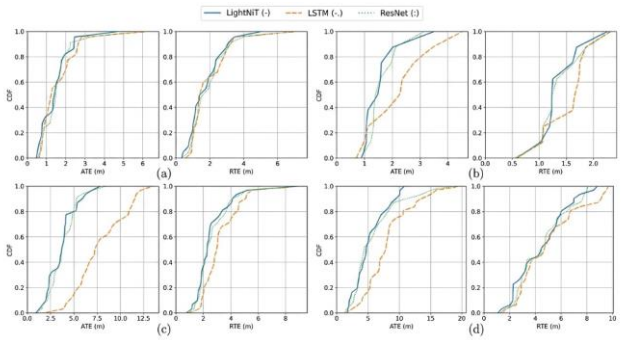

Fig.4. The CDF metrics of ATE and RTE on the test set are measured by different methods. The blue solid line, orange dotted line, and green dotted line correspond to LightNiT, LSTM, and ResNet networks, respectively.Figures a and b correspond to the RIDI and OXIOD datasets, and Figures c and d correspond to the visible and invisible sets of RONIN.

Figure 4 shows the CDF of LightNiT, RONIN-LSTM and RONIN-ResNet networks on the test set ATE and RTE. LightNiT shows high accuracy and stability in both ATE and RTE indicators, which is superior to the better ResNet and LSTM networks in RONIN. In short, our method can still achieve relatively optimal results while significantly improving the floating-point operation speed. This shows that LightNiT has strong potential in resource-constrained devices.

E. Ablation studies

TABLE III.PRIMARY METHOD POSITION EVALUATION

	Test subjects	Metric	ResNet	LightNiT-S	LightNiT-R	LightNiT
RIDI Dataset	Seen	ATE	1. 65	1. 6	1. 55	**1. 46**
		RTE	1. 93	1. 91	1. 88	**1. 77**
RONIN Dataset	Seen	ATE	3. 86	4. 02	3. 9	**3. 68**
		RTE	2. 74	2. 68	2. 73	**2. 61**
	Unseen	ATE	5. 98	5. 78	5. 98	**5. 26**
		RTE	4. 55	4. 62	4. 5	**4. 31**
OXOID Dataset	Seen	ATE	1. 66	1. 91	1. 75	**1. 6**
		RTE	1. 31	1. 45	1. 32	**1. 3**
Parameters(M)			4. 634	**0. 354**	0. 598	1. 086
FLOPs(M)			38. 25	**3. 82**	5. 47	8. 78
Inference Time (ms)			5. 5	**2. 01**	2. 91	5. 12

Table III provides a comparison of the accuracy, FLOPs, network parameters, and inference time of LightNiT and its variants and RONIN-ResNet networks under the three data sets,We roughed up the best results.We compare RONIN's best-performing ResNet network, LightNiT, and two other versions: LightNiT-S and LightNiT-R. In the LightNiT network, we superimpose the HybridAttnBlock module mentioned above into four layers, LightNiT-S only uses one layer, and LightNiT-R superimposes two layers. Compared with LightNiT, LightNiT-S and LightNiT-R reduce the number of network parameters by 67 % and 55 % respectively, FLOPs by 56 % and 38 % respectively, and the inference time is 1.5 times and 0.76 times faster. Among them, the average accuracy of LightNiT-S on the three datasets is slightly lower than that of RONIN-ResNet, but the number of parameters is significantly reduced. Compared with the ResNet network, the number of parameters is reduced by 92 %, FLOPs are reduced by 90 %, and the inference time is 1.7 times faster. The accuracy of LightNiT-R is comparable to that of RONIN-ResNet, the number of network parameters is reduced by 86 %, FLOPs are reduced by 86 %, and the inference time is 0.89 times faster.

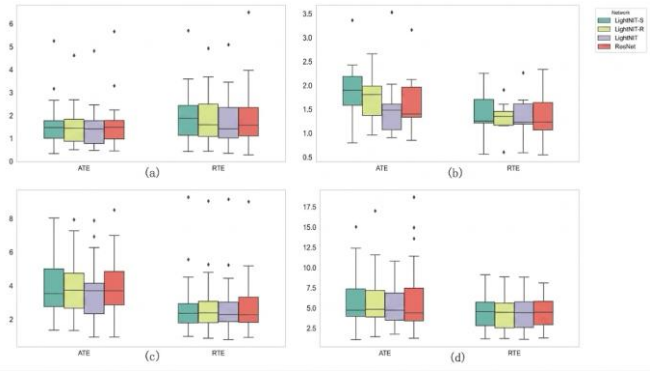

Fig.5. LightNiT and its variants and RONIN-ResNet box plots of ATE and RTE indicators in the selected test set. LightNiT-S, LightNiT-R, LightNiT and RONIN-ResNet networks are labeled with green, yellow, purple and magenta, respectively. Figures a and b correspond to the RIDI and OXIOD datasets, and Figures c and d correspond to the visible and invisible sets of RONIN.

Figure 5 shows the block diagram of ATE and RTE in the test set of LightNiT-S, LightNiT-R, LightNiT and RONIN-ResNet networks. It can be seen that the LightNiT series models have more concentrated data distribution and smaller ATE and RTE with the increase of the number of HybridAttnBlock modules mentioned above. Compared with ResNet, LightNiT network shows generally better performance than ResNet on all data sets, with the smallest ATE and RTE, and the data fluctuation is small, the data distribution is concentrated, and there are fewer outliers. The LightNiT-S and LightNiT-R networks are not as dense as ResNet in data distribution. However, due to the small number of outliers and the relatively stable network, the overall indicators are basically the same as ResNet.

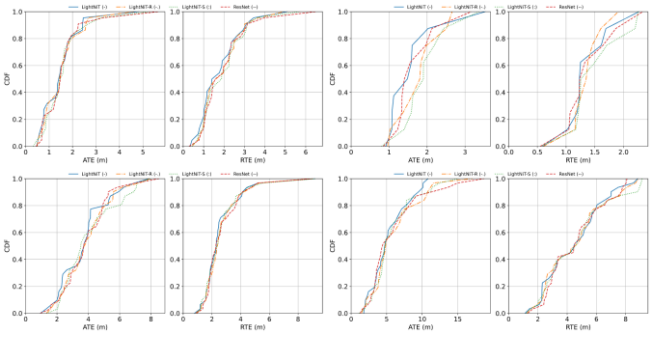

Fig.6. The CDF metrics of ATE and RTE on the test set of RONIN-ResNet, LightNiT and its variants are marked with blue solid, orange, green and red dashed lines, respectively, LightNiT, LightNiT-R, LightNiT-S and ResNet networks. Figures a and b correspond to the RIDI and OXIOD datasets, and Figures c and d correspond to the visible and invisible sets of RONIN.

Figure 6 shows the comparison of the cumulative distribution function (CDF) of the selected indicators of RONIN-ResNet, LightNiT and its variants in the test set. Compared with the ResNet network, LightNiT shows higher accuracy and robustness. LightNiT-R shows the same

accuracy level as ResNet. LightNiT-S shows slightly worse performance than ResNet from the perspective of random indicators.

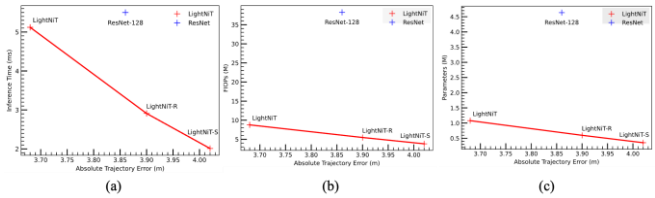

Fig.7. Performance metrics of RONIN-ResNet and LightNiT series networks in RONIN visible data sets are marked with blue and red crosses, respectively. Figures a, b and c are the comparison of inference time, FLOPs and network parameters with ATE, respectively.

Figure 7 shows the correspondence between the inference time, FLOPs and network parameters of RONIN-ResNet, LightNiT and their variants and the ATE of the RONIN visible test set. Further intuitively shows the advantages of the LightNiT network over the ResNet network in all performance indicators. In addition, it also shows that the LightNiT series network shows a positive correlation between the number of network parameters, FLOPs and inference time and the network effect to a certain extent, which further verifies the rationality of our network structure and the potential for further improvement of the network effect.

IV. DISCUSSION

The experimental results demonstrate that the proposed LTNIN achieves consistently better performance than existing deep learning-based inertial odometry methods, including RoNIN-ResNet, RoNIN-LSTM, and RoNIN-TCN, across all three datasets. The reduction in average absolute trajectory error (ATE) by 8%, 25.4%, and 27.2% indicates that the model effectively improves trajectory estimation accuracy.This improvement can be attributed to the lightweight Transformer-based architecture, which enhances the model's ability to capture temporal dependencies in inertial data while maintaining computational efficiency. Compared to conventional architectures, LTNIN strikes a better balance between accuracy and resource consumption, as evidenced by the reductions in FLOPs, number of parameters, and inference time.

The low computational cost and reduced drift make the model particularly suitable for deployment in resource-constrained environments, such as mobile edge devices. This suggests that LTNIN is not only an effective standalone inertial odometry solution but also a viable backup strategy in scenarios where computational resources are limited or visual inputs are unavailable.Considering the strong capability of Transformer architectures in multimodal learning, the proposed approach shows potential for extension to visual-inertial odometry systems. This indicates broader applicability beyond purely inertial settings.

However, several open challenges remain for practical real-world deployment. First, unlike conventional frameworks that integrate mature techniques such as the Extended Kalman Filter (EKF) or leverage multi-sensor inputs (e.g., magnetometers) to constrain cumulative drift in complicated or magnetically anomalous environments, the current LTNIN operates as a standalone, pure deep learning-based inertial odometer. While pure data-driven approaches excel at capturing complex non-linear motion patterns from raw IMU data, they can be sensitive to severe sensor noise fluctuations and hardware thermal drift in uncontrolled real-world settings. Furthermore, a well-known bottleneck for pure deep learning in pedestrian dead reckoning (PDR) is the reliability of reporting vertical (3D) displacement, as vertical acceleration signals are often heavily suppressed or masked by noise. Consequently, the current scope of this work is primarily optimized for high-precision 2D tracking and horizontal odometry. A promising and necessary direction for our future work is to explore a loose-coupled or tight-coupled integration of LTNIN with conventional EKF frameworks to enhance robustness against continuous sensor noise and extend the system's capability to reliable 3D trajectory estimation.

V. CONCLUSIONS

In this paper, we proposed LTNIN, a lightweight Transformer-based inertial odometer designed for speed regression and position estimation through integration. The proposed method improves upon LTNIN-Net by incorporating the LightViT architecture, resulting in enhanced performance and efficiency. Experimental results on multiple datasets demonstrate that LTNIN outperforms several state-of-the-art deep learning-based inertial odometry methods in terms of trajectory accuracy, while also achieving significant reductions in computational complexity and inference time. These characteristics make it well-suited for deployment in mobile and edge computing environments.

Our future work will follow three crucial directions. First, we will focus on the physical deployment of LTNIN on mobile terminals (e.g., smartphones and wearable devices) to systematically benchmark its runtime power consumption and real-time inference speed in uncontrolled, real-world environments. Second, we aim to design explicit noise-injection layers or adversarial training schemes during the optimization phase to further shield the lightweight Transformer topology against severe sensor noise and data inconsistency. Lastly, we plan to extend the standalone framework into a multi-sensor fusion system by incorporating complementary inputs (such as magnetometers or vision) and conventional kinematic constraints (e.g., EKF), expanding its applicability to complex human activity recognition (HAR) and comprehensive 3D navigation tasks.

ACKNOWLEDGMENT

The authors thank the editors and reviewers of this paper for their comments with which its quality was improved.

REFERENCES

[1] V. Carletti, A. Greco, A. Saggese, and M. Vento, "A smartphone-based system for detecting falls using anomaly detection," in International Conference on Image Analysis and Processing, Springer, 2017, pp. 490–499.

[2] M. Krichen, "Anomalies detection through smartphone sensors: A review," IEEE Sensors Journal, vol. 21, no. 6, pp. 7207–7217, 2021.

[3] Q. Wang et al., "Recent advances in pedestrian inertial navigation based on smartphone: A review," *IEEE Sensors Journal*, vol. 22, no. 23, pp. 22319–22343, 2022.

[4] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.
[5] H. Yan, Q. Shan, and Y. Furukawa, "RIDI: Robust IMU double integration," *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 621–636.
[6] C. Chen, X. Lu, A. Markham, and N. Trigoni, "Ionet: Learning to cure the curse of drift in inertial odometry," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.
[7] S. Herath, H. Yan, and Y. Furukawa, "Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3146–3152.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, DOI: 10.1162/neco.1997.9.8.1735.
[10] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," arXiv preprint arXiv:1803.01271, 2018.
[11] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[12] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," arXiv preprint arXiv:2010.11929, 2020.
[13] B.-K. Ruan, H.-H. Shuai, and W.-H. Cheng, "Vision transformers: state of the art and research challenges," arXiv preprint arXiv:2207.03041, 2022.
[14] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, pp. 4171–4186, 2019.
[16] H. Zhou et al., "Informer: Beyond efficient transformer for long sequence time-series forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11106–11115, 2021.
[17] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," *International Conference on Machine Learning, PMLR*, pp. 27268–27286, 2022.
[18] T. Brown et al., "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.
[19] B. Zhuang, J. Liu, Z. Pan, H. He, Y. Weng, and C. Shen, "A survey on efficient training of transformers," arXiv preprint arXiv:2302.01107, 2023.
[20] B. Rao, E. Kazemi, Y. Ding, D. M. Shila, F. M. Tucker, and L. Wang, "CTIN: Robust contextual transformer network for inertial navigation," *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5413–5421, 2022.
[21] S. Mehta and M. Rastegari, "Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer," arXiv preprint arXiv:2110.02178, 2021.
[22] S. Mehta and M. Rastegari, "Separable self-attention for mobile vision transformers," arXiv preprint arXiv:2206.02680, 2022.
[23] T. Huang, L. Huang, S. You, F. Wang, C. Qian, and C. Xu, "Lightvit: Towards light-weight convolution-free vision transformers," arXiv preprint arXiv:2207.05557, 2022.
[24] K. Wu et al., "Tinyvit: Fast pretraining distillation for small vision transformers," European Conference on Computer Vision, pp. 68–85, 2022.
[25] S. Ek, F. Portet, and P. Lalanda, "Lightweight transformers for human activity recognition on mobile devices," arXiv preprint arXiv:2209.11750, 2022.
[26] S. S. Saha, S. S. Sandha, L. A. Garcia, and M. Srivastava, "Tinyodom: Hardware-aware efficient neural inertial navigation," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–32, 2022.
[27] T. Brovko, A. Chugunov, A. Malyshev, I. Korogodin, N. Petukhov, and O. Glukhov, "Complex Kalman filter algorithm for smartphone-based indoor UWB/INS navigation systems," *2021 Rural Symposium on Biomedical Engineering, Radioelectronics and Information Technology*, pp. 0280–0284, 2021.
[28] A. H. Pourmina, M. M. Alizadeh, and H. Schuh, "Decimeter-level accuracy for smartphone real-time kinematic positioning implementing a robust Kalman filter approach and inertial navigation system infusion in complex urban environments," *Sensors*, vol. 24, no. 18, p. 5907, 2024.
[29] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, "Oxiod: The dataset for deep inertial odometry," arXiv preprint arXiv:1809.07491, 2018.