Faster than Light: An Embedded-Efficient Matching Model with ReLU Linear Attention

Ziang Wang[1], Tao He[1], Wei Cui[1], Yu Duan[1], Kaimin Sun[1], Yunhao Miao[2]

[1] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan, China - (2023286190106, hetao828, cuiwei, 2023106190026, sunkm)@whu.edu.cn
[2] North Automatic Control Technology Institute. Taiyuan, China - 1023463276@qq.com

Keywords: Image matching, Linear attention, Graph neural network, Embedded vision.

Abstract

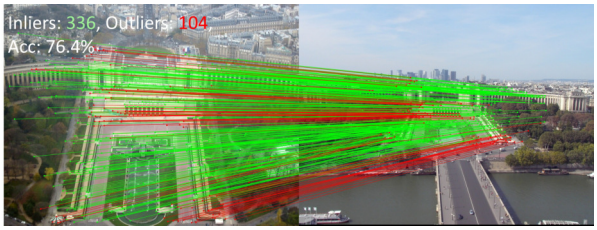
Deep learning-based image matching faces a critical challenge when deployed on computationally constrained embedded aerial devices. Transformer-based architectures, particularly the scaled dot-product attention mechanism, incur high computational costs that limit inference speed for real-time applications. To address this bottleneck, we propose FastGlue, a sparse feature matching algorithm that adapts the LightGlue architecture through two targeted modifications: replacing the scaled dot-product attention with a ReLU-based linear attention module, and reducing the depth of the graph neural network. These changes reduce computational complexity while maintaining competitive matching performance. Evaluations on HPatches and MegaDepth-1500 benchmarks show that FastGlue achieves accuracy comparable to LightGlue while improving inference speed—from 20.05 ms to 17.05 ms on GPU, and from 840.45 ms to 665.85 ms on an RK3588 embedded CPU. Our work demonstrates that targeted architectural simplifications can yield meaningful efficiency gains for deep learning-based feature matching on resource-constrained platforms.

1. Introduction

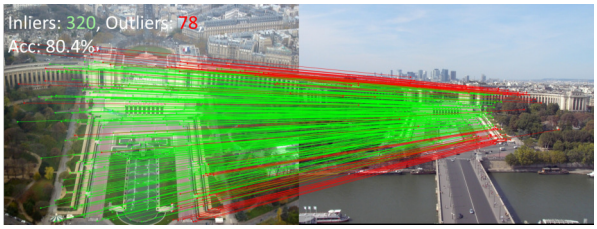
In the field of drone remote sensing, image matching models play a critical role. These models are used to register images of the same area captured at different times or by different sensors, enabling applications such as surface change detection, target annotation, and object recognition. They are widely employed in disaster monitoring, environmental protection, land use surveys, and many other domains.

Image matching techniques have evolved from traditional handcrafted feature-based methods to modern deep learning-based approaches. While traditional methods such as SIFT (Lowe, 2004) and ORB (Rublee et al., 2011) laid the foundation for feature extraction and matching, their reliance on manually designed features inherently limits robustness and generalization in complex scenarios. The emergence of deep learning has markedly improved matching accuracy, leading to powerful models such as SuperGlue (Sarlin et al., 2020), LoFTR (Sun et al., 2021), DKM (Edstedt et al., 2022), and LightGlue (Lindenberger et al., 2023). These methods have demonstrated clear advantages over traditional techniques in terms of both precision and inference speed on standard hardware, establishing deep learning as the prevailing paradigm in this domain.

However, as research progresses, a critical limitation has become apparent: most leading deep learning-based matchers rely heavily on Transformer-based backbones, which depend on computationally intensive matrix operations. While effective on high-performance workstations or servers, such architectures encounter severe efficiency bottlenecks when deployed on computationally constrained platforms, such as drone-embedded systems. The resulting inference delays hinder their applicability in real-time tasks including visual navigation and target localization, underscoring the need for matching models that reconcile high accuracy with embedded-level efficiency. Among existing methods, LightGlue achieves a favorable balance between speed and accuracy on standard hardware, yet its inference time on embedded CPUs—approximately 840 ms on an RK3588 platform—remains too high for real-time aerial applications.



(a) LightGlue



(b) FastGlue

Figure 1. Qualitative comparison of matching performance between LightGlue and our proposed method called FastGlue on outdoor visual localization, given exactly the same keypoints and initial descriptors provided by SuperPoint under the same threshold parameters.

To address this limitation, this paper proposes a sparse feature matching method that adapts the LightGlue architecture for improved efficiency on embedded CPUs. Our approach builds upon the LightGlue framework—which itself extends SuperGlue with confidence-based early termination—and introduces two modifications to reduce computational cost without compromising matching accuracy. First, we replace the scaled dot-product self-attention in the graph neural network backbone with ReLU linear attention, drawing on the observation by Wortsman et al. (Wortsman et al., 2023) that linear atten-

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-2-2026-45-2026

tion variants can effectively reduce complexity while maintaining representational capacity. Second, we reduce the depth of the graph neural network and evaluate the impact of this design choice on the trade-off between inference speed and matching accuracy. In contrast to prior work that primarily optimizes for GPU inference, our modifications are specifically motivated by the constraints of embedded CPU platforms.

Specifically, the main contributions of this paper are as follows:

(1) We propose FastGlue, a lightweight sparse feature matching model designed for improved inference efficiency on embedded CPUs. The model adapts the LightGlue architecture through targeted simplifications tailored to resource-constrained platforms.

(2) We introduce two modifications to the LightGlue backbone: replacement of scaled dot-product attention with ReLU linear attention, and reduction of graph neural network depth. We analyze the impact of each modification on the speed-accuracy trade-off.

(3) We evaluate FastGlue on HPatches and MegaDepth-1500 benchmarks, comparing its performance against LightGlue and other state-of-the-art methods. Experimental results show that our modifications reduce inference time from 20.05 ms to 17.05 ms on GPU and from 840.45 ms to 665.85 ms on an RK3588 embedded CPU, with comparable matching accuracy.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 presents the proposed methodology in detail. Experimental results are reported and discussed in Section 4. Finally, Section 5 concludes the paper and outlines potential directions for future research.

2. Related Work

Image matching algorithms serve as a key technology in remote sensing image processing. With advancements in remote sensing imaging technologies and the continuous expansion of image data scales, image matching algorithms have gradually become an active research focus in remote sensing applications. Existing image matching algorithms can be broadly categorized into traditional methods and deep learning-based approaches. The following section elaborates on the research status of these two categories.

2.1 Traditional Matching Methods

Traditional image matching has long relied on handcrafted feature extraction and description pipelines, with SIFT establishing the canonical framework for scale and rotation-invariant matching. While subsequent algorithms like SURF (Gool, 2008) and ORB made significant efficiency improvements.ORB in particular achieved notable speed through binary features, and these methods share fundamental limitations. Their reliance on manually designed features inherently restricts their adaptability and robustness when facing complex real-world variations such as significant illumination changes, noisy conditions, or non-rigid transformations. This inherent lack of generalization, coupled with their sequential and often iterative processing stages, ultimately constrains both their performance in challenging environments and their efficiency for large-scale or real-time applications, thereby motivating the shift towards data-driven, deep learning-based approaches.

2.2 Deep Learning–Based Matching Methods

While deep learning has revolutionized image matching, current methodologies reveal distinct limitations when considered for real-world embedded applications. Initial breakthroughs like SuperPoint (DeTone et al., 2017) and D2-Net (Dusmanu et al., 2019) demonstrated the power of learned features but operated as disjoint detection and description modules, lacking a holistic matching strategy. The advent of powerful matchers like SuperGlue and LoFTR marked a significant leap by modeling matches within an end-to-end optimization framework, leveraging attention mechanisms and dense correlations to achieve impressive accuracy.

Building upon these foundations, subsequent works have explored various directions for improvement. XoFTR (Önder Tuzcuoğlu et al., 2024) extends LoFTR to cross-modal matching scenarios, while LightGlue introduces confidence-based early termination to accelerate inference. More recently, MambaGlue and MapGlue (Wu et al., 2025) have adapted the SuperGlue architecture with alternative backbone designs for improved efficiency, while RoMa (Edstedt et al., 2024) leverages foundation model features to achieve state-of-the-art accuracy on challenging benchmarks.

However, this performance comes at a substantial computational cost. The core of these state-of-the-art models often relies on heavy Transformer-based architectures or dense attention mechanisms, which are notoriously resource-intensive. While subsequent improvements such as LightGlue have made commendable strides in efficiency, they often achieve this through architectural tweaks or model compression without fundamentally addressing the computational bottleneck of the core matching operator. Consequently, these models still struggle with the stringent latency and power constraints of real-time deployment on embedded aerial platforms, where computational resources are severely limited. This critical gap between desktop-level accuracy and on-device efficiency underscores the need for a fundamentally more efficient matching paradigm.

3. Methods

The proposed FastGlue model comprises a graph neural network and an optimal matching layer, with its detailed network structure illustrated in Figure 2. The network processes input images A and B using the SuperPoint algorithm to extract initial feature points and descriptors (p^A, d^A) and (p^B, d^B), which serve as inputs to the feature matching network.

The key innovation lies in our redesigned attention in the graph neural network. We replace the conventional Softmax operator with ReLU activation in self-attention modules, reducing the computational complexity from $O(n^2)$ to $O(n)$ while maintaining the functionality of attention. This modification significantly decreases the computational load during both training and inference. Furthermore, we employ a lightweight graph attention architecture with only 5 layers, substantially shallower than the 9-layer structures used in SuperGlue and LightGlue. The processed features undergo iterative refinement through the graph attention network, generating optimized descriptors d_n^A and d_n^B. A confidence classifier subsequently evaluates each feature point, pruning both non-matchable points and those with low confidence scores. This pruning operation occurs between graph attention layers, with the network either proceeding to the next layer or terminating early based on two

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-2-2026-45-2026

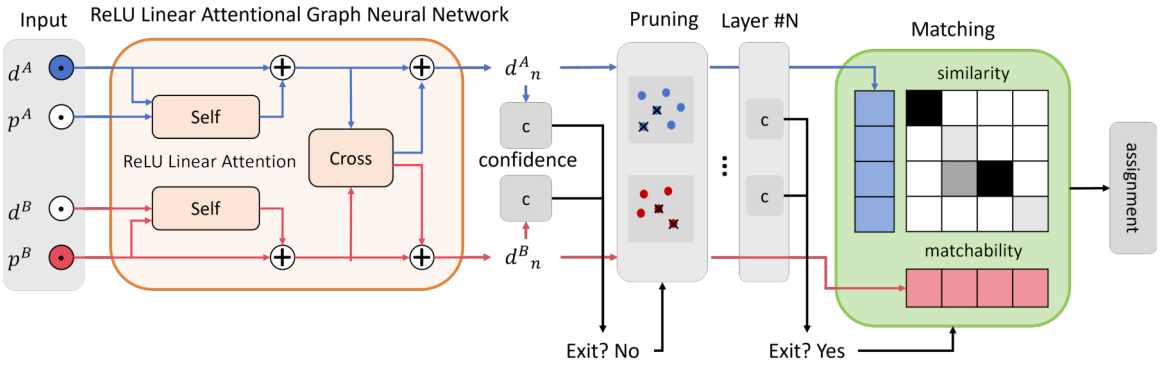


Figure 2. Overview of the proposed feature matching model called FastGlue.

conditions: when the 5-layer depth is reached, or when the number of high-confidence feature points meets the predefined threshold. Finally, the refined descriptors are fed into the optimal matching layer, where an optimization algorithm iteratively processes the similarity matrix to produce the final matching results.

3.1 ReLU Linear Attentional Graph Neural Network

Taking the feature descriptors d^A and their corresponding keypoints p^A from image A as an example, these descriptors are processed through a self-attention followed by a residual connection that adds the processed output to the original descriptors, yielding the self-attention enhanced vectors d^A_{self}:

$$d^A_{self} = self(d^A, p^A) + d^A \tag{1}$$

In (1), $self(.)$ denotes the computational process of self-attention. The self-attention vectors d^A_{self} encapsulate not only the implicit representations of texture, orientation, and other characteristics from the original descriptors, but also incorporate positional information learned through the encoding of keypoints. These enhanced representations are subsequently processed via a cross-attention module to generate the cross-attention vectors d^A_{self}:

$$d^A_{cross}, d^A_{cross} = cross(d^A_{self}, d^B_{self}) \tag{2}$$

In (2), $cross(.)$ denotes the computational process of cross-attention, d^B_{self} denotes the self-attention vectors obtained by processing the keypoints d^B and descriptors p^B of image B through the self-attention. Through the cross-attention vectors, the network incorporates the implicit information representation carried by the self-attention enhanced vectors d^B_{self} of the other image (Image B), which is then combined with the original self-attention vectors through a residual connection to produce the output d^A_n of this structural block:

$$d^A_n = d^A_{self} + d^A_{cross} \tag{3}$$

(1), (2) and (3) constitute a single layer of the graph attention network, whose output serves d^A_n and d^B_n as the input feature descriptors for the subsequent layer. Typically, such graph neural networks require N repeated structural units to iteratively refine the feature descriptors. Conventionally, N is set to 9, meaning the graph attention module undergoes nine computational iterations, requiring a total of 27 multi-head attention operations. While repeated attention-based encoding enhances the robustness of feature descriptors, excessive attention significantly degrade both training and inference speed. Therefore, appropriately reducing the number of attention operations can effectively accelerate the matching network.

Investigating the impact of graph neural network depth on both inference speed and accuracy is crucial. In FastGlue, we set the maximum number of graph attention layers to 5. The effectiveness of this configuration will be validated through experiments in Section 4.5.2.

3.2 ReLU Linear Attention

Although models based on the Transformer architecture have demonstrated remarkable performance advantages in various visual tasks, empirical studies indicate that they face significant computational efficiency bottlenecks when processing long-sequence inputs (such as when the number of feature points involved in matching substantially exceeds the channel dimension of their feature descriptors.). For the Scaled Dot-Product Attention, its core lies in the computation of three matrices: Q, K, and V:

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4}$$

The $Attention$ means scaled dot-product attention, and the d_k equals to the channel dimension of the feature descriptor. We employ SuperPoint as our feature extraction network, consequently the d_k is 256. The scaled dot-product attention computation process consists of the following three steps:

First, the similarity matrix Sim is computed as:

$$Sim = QK^T \tag{5}$$

Each row of Q (dimension: $R^{1\times d}$) is multiplied with each column of K^T (dimension: $R^{d\times 1}$) via dot product, yielding a scalar value. Each element requires d multiplications and $(d-1)$ additions, with $n \times n$ elements in total. Thus, the computational complexity of this step is:

$$O((n \times n) \times (d + d - 1)) = O(n^2 d) \tag{6}$$

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-2-2026-45-2026 | © Author(s) 2026. CC BY 4.0 License.

Then the similarity matrix is normalized as:

$$A = Softmax(Sim) \in R^{n \times n} \tag{7}$$

Where A is the normalized similarity matrix. This step computes exponentials and performs normalization for each of the n elements across all n rows. With $O(n)$ operations per row and n rows in total, the complexity becomes:

$$O(n \times n) = O(n^2) \tag{8}$$

Finally, the attention output is obtained by:

$$Attention(Q, K, V) = A \cdot V \in R^{n \times d} \tag{9}$$

Matrix multiplication between A and V involves $n \times d$ elements, each requiring n multiplications and $(n-1)$ additions. Therefore, the complexity of this step is:

$$O((n \times d) \times n) = O(n^d) \tag{10}$$

In summary, the overall computational complexity of the scaled dot-product attention is derived as:

$$O(n^d) + O(n^) + O(n^d) = O(n^d) \tag{11}$$

In typical feature matching scenarios, the sequence length n (the number of image feature keypoints, typically n = 1024 or 2048) is often much larger than d. This dimensional disparity causes standard attention to encounter significant computational bottlenecks. Based on this, it is necessary to construct a novel attention computation method with linear time complexity $O(n)$ to achieve breakthrough improvements in algorithmic efficiency while maintaining matching accuracy.

To address this issue, the linear attention (Katharopoulos et al., 2020) proposes substituting the softmax normalization function with the ELU function, thereby reducing the computational complexity of attention from $O(n^2 d_k)$ to $O(nd_k^2)$. However, the ELU function itself introduces considerable computational overhead. Therefore, we adopt a simpler ReLU function as an alternative to softmax and design a ReLU linear attention. ReLU linear attention network structure illustrated in Figure 3.

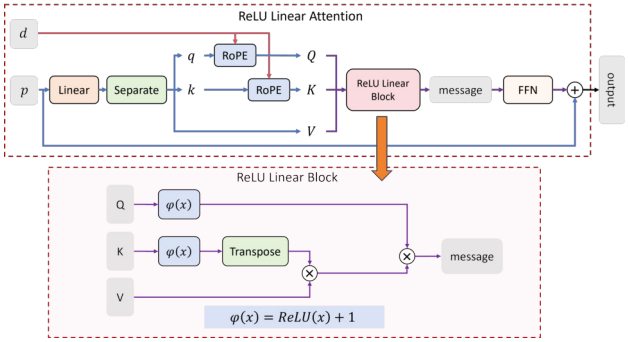


Figure 3. ReLU linear attention network.

Specifically, we need to derive the Q, K, and V from the input keypoints d and descriptors p first:

$$W_{qkv} = Linear(p) \tag{12}$$

In (12), $Linear$ denotes a linear layer that transforms the channel dimension of the output vector W_{qkv} to three times the channel dimension of the input descriptors. Then, the Q, K, and V matrices can be separated from W_{qkv}:

$$q, k, V = Separate(W_{qkv}) \tag{13}$$

In (13), $Separate$ denotes a series of operations where W_{qkv} undergoes unflatten and transpose procedures, after which the resulting feature vector is split along the channel dimension to obtain the final q, k, and V matrices. Subsequently, we incorporate rotary position encoding ($RoPE$) into matrices q and k, enabling the model to perceive positional information carried by keypoints in the sequence:

$$Q = RoPE(q, d); K = RoPE(q, d) \tag{14}$$

Using the Q, K, and V matrices obtained through the above formulation, we can compute the $Message$, which is the output of the ReLU linear block as follows:

$$Message = \frac{\varphi(Q)(\varphi(K)^T V)}{\varphi(Q)(\varphi(K)^T 1_n)} \tag{15}$$

where 1_n denotes an n-dimensional vector of all ones, and the denominator performs the normalization operation along the sequence dimension, and $\varphi(x)$ is defined by (16):

$$\varphi(x) = ReLU(x) + 1 \tag{16}$$

where x is just a parameter. In the ReLU linear attention formulation corresponding to (15), we first compute $K^T V$ instead of QK^T, yielding an intermediate matrix B:

$$B = K^T \cdot V \in R^{d \times d} \tag{17}$$

Subsequently, we left-multiply Q with B to obtain the final output $Attention_L$:

$$Attention_L(Q, K, V) = Q \cdot B \in R^{n \times d} \tag{18}$$

The overall computational complexity of this process is:

$$O(n \times d \times d) = O(nd^2) \tag{19}$$

This approach reduces the complexity with respect to sequence length to linear $O(n)$, thereby achieving significantly higher computational efficiency in feature matching tasks. The final output of the self-attention network can then be obtained as follow:

$$Ouput = FFN(Message) + p \tag{20}$$

where FFN is a feed-forward neural network (FFN), and $Ouput$ is the final output of the ReLU linear attention network.

3.3 Pruning

To eliminate unmatchable keypoints from the images, we incorporate a confidence classifier into the network for pruning both unmatchable points and those with low confidence. Specifically, the descriptors d output by each graph neural network layer are processed through a linear layer followed by a Sigmoid activation function to obtain the confidence scores C_{token} for the descriptors:

$$C_{token}(d) = Sigmoid(Linear(d)) \tag{21}$$

These confidence scores are then used to dynamically prune low-confidence keypoints during the matching process, with the pruning threshold T being adaptively determined based on the distribution of confidence values in each matching pair. The threshold T for descriptor confidence is calculated using the following:

$$T = 0.8 + 0.1e^{(-4l/L)} \tag{22}$$

where L represents the total number of layers in the graph attention network and l indicates the current layer index. Descriptors with confidence scores exceeding this threshold are classified as reliable. When the proportion of such reliable descriptors reaches 95% of the total (a threshold set to ensure matching accuracy), we consider the network output sufficient for subsequent optimal matching layers and may thus terminate the graph attention network early. If the number of qualified descriptors is insufficient, low-confidence descriptors are pruned and the remaining descriptors proceed to the next graph attention layer for further refinement, thereby accelerating the inference process.

3.4 Optimal Matching Layer

The optimal matching layer iteratively refines the matching results by computing a similarity matrix and a matching probability matrix. In essence, this layer transforms the matching problem into an optimization task, seeking to maximize the sum of element-wise products between the two matrices to obtain the optimal matching probability matrix. The similarity matrix $S_{(A,B)}$ is computed as the inner product of the d_n^A and d_n^B output from ReLU linear attentional graph neural network:

$$S_{(A,B)} = d_n^A \cdot d_n^B \in R^{M \times N} \tag{23}$$

where M and N denote the number of feature points in image A and image B, respectively. The matching probability matrix $P_{(A,B)}$ is then derived by solving the optimization problem that maximizes the Frobenius inner product $S_{(A,B)} \cdot P_{(A,B)}$.

To facilitate subsequent computations, we aim to constrain the elements of the final matching probability matrix $P_{(A,B)}$ to the range [0, 1]. However, the similarity matrix $S_{(A,B)}$ generated by the feature matching network typically contains values spanning a considerably wider numerical range. Therefore, we first project the similarity matrix into logarithmic space to initialize the probability matrix:

$$P_{(A,B)} = e^{S_{(A,B)}} \tag{24}$$

Subsequently, we apply the Sinkhorn algorithm to iteratively normalize each row and column of $P_{(A,B)}$ until convergence, ensuring all rows and columns sum to unity. The final matching results are determined by identifying, for each row a in the converged matrix, the column b containing the maximum value, indicating that feature point a in image A corresponds to feature point b in image B.

4. Experiments

4.1 Experimental Datasets and Setups

We evaluate FastGlue on two tasks: homography estimation and relative pose estimation. For homography estimation, we use the HPatches dataset (Balntas et al., 2017) which provides image pairs with ground-truth homographies under challenging conditions including illumination and viewpoint changes. For relative pose estimation, we use the MegaDepth-1500 dataset (Li and Snavely, 2018), which contains 1,500 image pairs from outdoor scenes with geometric annotations. Additionally, we benchmark the average inference time of each model on the MegaDepth-1500 dataset to evaluate their speed across different hardware platforms.

We compare FastGlue against three deep learning-based feature matchers: SuperGlue, LightGlue, and LoFTR. SuperGlue and LightGlue are sparse matchers that operate on keypoints extracted by a detector (SuperPoint in our experiments), while LoFTR is a dense matcher that establishes matches without explicit keypoint detection. We include LoFTR as a representative dense method to illustrate the performance-efficiency trade-off across matching paradigms. In addition, we add a traditional baseline combining ORB features with brute-force matching to provide a reference for the practical gains of deep learning-based methods on embedded hardware. Recent dense matchers such as RoMa are not included, as they operate on a different paradigm from the sparse methods that are the primary focus of this work.

For training, we pre-train FastGlue, SuperGlue, and LightGlue on RevisitOP1M (Radenovi et al., 2018) for 40 epochs and fine-tune on MegaDepth (Li and Snavely, 2018) for 50 epochs. All three models are trained on a workstation equipped with eight RTX 4080 GPUs. We use official pre-trained weights for LoFTR. The ORB baseline requires no training. For evaluation, we use SuperPoint with a maximum of 1024 keypoints for all sparse matchers (FastGlue, SuperGlue, LightGlue) to ensure a fair comparison. Evaluations are conducted on two platforms: the training workstation (with an RTX 4080 GPU) and an embedded board featuring an RK3588 CPU, to assess performance across both high-performance and resource-constrained environments.

4.2 Homography Estimation

We evaluated homography estimation accuracy using both robust (RANSAC (Fischler and Bolles, 1981)) and non-robust

(Weighted DLT (Larsson and contributors, 2020)) estimators. RANSAC effectively handles outliers through random sampling, whereas DLT computes the homography directly but is more susceptible to errors with noisy data. Evaluation metrics include the Area Under the Curve (AUC) for cumulative mean reprojection errors at 1 px and 5 px thresholds, as well as precision at error thresholds of 1 px and 5 px, and recall at error thresholds of 1 px and 3 px. The tests are conducted on a GPU, with results on the HPatches dataset presented in Table 1.

models	Precision		Recall		AUC–RANSAC(%)		AUC-DLT(%)	
	@1px	@3px	@1px	@5px	@1px	@5px	@1px	@5px
ORB	0.0	5.3	2.0	3.1	2.0	2.7	0.0	0.0
LoFTR	-	92.7	-	-	41.5	78.8	38.5	70.6
SuperGlue	32.8	92.7	64.1	87.4	32.1	67.5	32.0	75.6
LightGlue	33.2	**94.1**	**65.6**	**94.4**	33.5	**68.4**	32.2	75.7
Ours.	**33.3**	93.7	64.3	93.5	**33.7**	67.7	**33.4**	**76.0**

Table 1. Homography estimation accuracy of image matching models on the HPatches dataset.

Table 1 presents the homography estimation results. Among sparse matchers, FastGlue achieves a precision of 33.3% at the 1 px threshold, slightly higher than LightGlue's 33.2% and SuperGlue's 32.8%. At the 3 px threshold, FastGlue (93.7%) trails LightGlue (94.1%) by 0.4 percentage points, while remaining above SuperGlue (92.7%). For recall at 1 px, FastGlue (64.3%) falls between LightGlue (65.6%) and SuperGlue (64.1%); at 5 px, it reaches 93.5%, compared to LightGlue's 94.4% and SuperGlue's 87.4%.

We attribute FastGlue's strong performance at stricter thresholds (1 px) to the ReLU linear attention mechanism, which may produce more localized and precise matching decisions compared to the scaled dot-product attention used in LightGlue. The slight drop at the 3 px threshold, however, suggests that the shallower network depth may limit the model's ability to capture broader contextual information that becomes more important when allowing larger reprojection errors. This trade-off—maintaining or improving precision at stricter thresholds while slightly underperforming at looser ones—reflects the design choices made to prioritize efficiency.

When using RANSAC for homography estimation, FastGlue achieves an AUC of 33.7% at the 1 px threshold, marginally higher than LightGlue's 33.5% and SuperGlue's 32.1%. Under the DLT estimator, FastGlue shows a more consistent advantage, with AUC values of 33.4% at 1 px and 76.0% at 5 px, compared to LightGlue's 32.2% and 75.7%. This difference between estimators may indicate that matches produced by FastGlue contain fewer gross outliers than those from LightGlue, making it less dependent on RANSAC's robust sampling to achieve accurate homography estimates. In contrast, LightGlue's higher recall at larger thresholds may come at the cost of introducing more outliers that are effectively handled by RANSAC but degrade DLT performance. The ORB baseline performs substantially lower across all metrics, highlighting the performance gap between traditional and learning-based approaches on this benchmark. LoFTR, as a dense matcher, achieves strong results particularly in recall and AUC, but these gains come at a higher computational cost that we analyze in Section 4.4.

4.3 Relative Pose Estimation

For relative pose estimation on the MegaDepth-1500 dataset, we computed the essential matrix using both RANSAC and LO-RANSAC with LM refinement (Larsson and contributors, 2020). The pose error for each image pair was measured as the maximum angular error in rotation, and we report the Area Under the Curve (AUC) at thresholds of 5°, 10°, and 20°. The matching accuracy of each model under this setting is summarized in Table 2.

models	AUC-RANSAC(%)			AUC-LO-RANSAC(%)		
	5°	10°	20°	5°	10°	20°
ORB	0.1	0.1	0.5	0.0	0.1	0.2
LoFTR	52.8	69.2	81.2	66.4	78.6	86.5
SuperGlue	46.7	63.9	77.4	61.1	75.1	85.1
LightGlue	**49.5**	**66.5**	**79.3**	64.7	77.0	86.6
Ours.	47.8	65.3	78.3	**64.9**	**77.5**	**86.6**

Table 2. Pose estimation accuracy of image matching models on the MegaDepth-1500 dataset.

Table 2 presents the pose estimation results. When using RANSAC, FastGlue achieves AUC scores of 47.8%, 65.3%, and 78.3% at the 5°, 10°, and 20° thresholds, respectively. These values fall between SuperGlue (46.7%, 63.9%, 77.4%) and LightGlue (49.5%, 66.5%, 79.3%), with FastGlue trailing LightGlue by margins of 1.7, 1.2, and 1.0 percentage points. Under LO-RANSAC, which performs iterative refinement of the estimated essential matrix, FastGlue achieves the highest AUC among all compared methods across all three thresholds: 64.9% at 5°, 77.5% at 10°, and 86.6% at 20°, slightly surpassing LightGlue (64.7%, 77.0%, 86.6%) and LoFTR (66.4%, 78.6%, 86.5%) at the 20° threshold.

The difference in relative performance between the two estimators offers insight into the characteristics of matches produced by each model. Under standard RANSAC, LightGlue maintains a consistent advantage over FastGlue across all thresholds. However, when LO-RANSAC applies additional refinement, FastGlue matches or slightly exceeds LightGlue. This pattern suggests that matches from FastGlue may be of comparable quality but require more precise optimization to fully realize their potential—potentially due to the shallower network depth or the ReLU attention mechanism producing matches that are locally accurate but less geometrically consistent across the broader scene. Conversely, LightGlue's matches may be more robust to standard RANSAC sampling but offer less room for improvement through iterative refinement. As with homography estimation, the ORB baseline performs poorly across all settings, reinforcing the substantial gap between traditional and learning-based approaches. LoFTR, as a dense matcher, achieves strong results particularly at stricter thresholds under LO-RANSAC (66.4% at 5°), but this performance comes with higher computational cost, which we analyze in Section 4.4.

Overall, the results on MegaDepth-1500 are consistent with those on HPatches: FastGlue maintains competitive accuracy with LightGlue, with differences that depend on the specific estimator used. The trade-off between precision at strict thresholds and robustness under different estimation procedures reflects the design choices made to prioritize inference efficiency on embedded hardware.

4.4 Performance of Time Efficiency

To evaluate the inference efficiency of FastGlue, we measured its runtime on the MegaDepth-1500 dataset across different platforms. Due to the slow inference on the RK3588 chip, we converted SuperGlue, LightGlue, and FastGlue into ONNX

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-2-2026-45-2026

format and recorded their average inference time. The results are presented in Table 3.

Platform	models	GNN(ms)	Matching(ms)	time (ms)
GPU	LoFTR	–	–	181.12
GPU	SuperGlue	21.38	19.47	40.85
GPU	LightGlue	18.16	**1.82**	19.98
GPU	Ours.	**9.77**	7.28	**17.05**
CPU	LoFTR	–	–	36819.18
CPU	SuperGlue	11550.41	2730.73	14281.14
CPU	LightGlue	8613.53	**175.58**	8789.12
CPU	Ours.	**6282.27**	178.79	**6461.07**
CPU	SuperGlue$_{onnx}$	–	–	894.48
CPU	LightGlue$_{onnx}$	–	–	822.43
CPU	Ours.$_{onnx}$	–	–	**665.85**

Table 3. Inference time of matching models on MegaDepth-1500 dataset across platforms.

As shown in Table 3, the runtime characteristics differ substantially between GPU and CPU platforms. On GPU, LightGlue achieves fast inference (19.98 ms), with most time spent in the GNN (18.16 ms) rather than the matching layer (1.82 ms). SuperGlue shows a different distribution, with comparable time in GNN (21.38 ms) and matching (19.47 ms). FastGlue reduces GNN time by nearly 50% compared to LightGlue (9.77 ms vs. 18.16 ms), but incurs higher matching layer cost (7.28 ms vs. 1.82 ms). This trade-off arises because the matching layer's optimal transport algorithm relies on feature quality: LightGlue's deeper GNN produces more refined features that enable rapid convergence, while FastGlue's shallower GNN—though faster to compute—yields less refined features, requiring more iterations in the matching stage. Overall, FastGlue achieves a modest speedup over LightGlue (17.05 ms vs. 19.98 ms) on GPU.

On the embedded CPU RK3588, the performance landscape shifts significantly. LightGlue's GNN becomes the dominant cost (8613.53 ms), while its matching layer remains efficient (175.58 ms). FastGlue reduces GNN time by approximately 27% (6282.27 ms vs. 8613.53 ms), with comparable matching layer overhead (178.79 ms vs. 175.58 ms). This results in a total inference time of 6461.07 ms for FastGlue, compared to 8789.12 ms for LightGlue—a reduction of about 2.3 seconds. The different relative performance between GPU and CPU reflects the fact that GPU acceleration disproportionately benefits the matrix operations in deeper GNNs, narrowing the gap between the two models on GPU. On CPU, where memory access patterns and sequential execution dominate, the reduction in GNN depth yields more tangible gains. The dense matcher LoFTR is the slowest overall on both platforms, taking 181.12 ms on GPU and over 36 seconds on CPU, highlighting the computational cost of dense feature processing. After conversion to ONNX format for deployment, FastGlue maintains a consistent speed advantage on the embedded CPU, with inference times of 665.85 ms compared to LightGlue's 822.43 ms and SuperGlue's 894.48 ms.

4.5 Ablation Study

4.5.1 ReLU Linear Attention: To validate the effectiveness of our proposed ReLU linear attention, we compare the performance and speed of matching models using different attention mechanisms. All models share the same FastGlue architecture, with the only variation being the attention module: scaled dot-product attention (SDPA), standard linear attention (Linear), or our ReLU linear attention (Ours.). Experiments are conducted on the MegaDepth-1500 dataset.

Attention	AUC-RANSAC(%)			AUC-LO-RANSAC(%)			GPU (ms)	CPU(ms)
	5°	10°	20°	5°	10°	20°		
SDPA	47.7	65.0	78.1	**65.3**	**78.0**	**86.9**	17.48	6859.35
Linear	46.9	64.6	78.3	64.7	77.7	86.7	17.42	6651.09
Ours.	**47.8**	**65.3**	**78.3**	64.9	77.5	86.6	**17.05**	**6461.07**

Table 4. Pose estimation accuracy and inference time of models with different attentions on the MegaDepth-1500 dataset.

Table 4 shows that the three attention mechanisms achieve generally consistent performance across metrics, though standard linear attention underperforms relatively. Despite its simplified mapping, ReLU linear attention delivers notably strong results in pose estimation: it achieves the highest accuracy with RANSAC and remains competitive with scaled dot-product attention when using LO-RANSAC. In terms of inference speed, models equipped with ReLU linear attention achieve the shortest average runtime on both GPU and RK3588 platforms. This indicates that ReLU linear attention offers a superior balance between efficiency and accuracy.

4.5.2 The Depth of the Graph Neural Network: To validate the design choice of using a 5-layer ReLU linear attentional graph neural network, we conduct experiments with varying GNN depths while keeping all other components of FastGlue unchanged. Models with 3, 5, 7, and 9 graph network layers are evaluated on the MegaDepth-1500 dataset in terms of pose estimation accuracy and inference speed across different platforms.

Layers	AUC-RANSAC(%)			AUC-LO-RANSAC(%)			GPU (ms)	CPU(ms)
	5°	10°	20°	5°	10°	20°		
9	**49.0**	66.2	**79.5**	**65.2**	**78.0**	**87.1**	28.87	11366.01
7	48.7	**66.5**	78.6	65.1	78.0	87.1	22.94	8626.27
5	47.8	65.3	78.3	64.9	77.5	86.6	17.05	6461.07
3	44.6	62.3	76.3	63.2	76.2	85.5	**11.18**	**4060.32**

Table 5. Pose estimation accuracy and inference time of models with different GNN depths on the MegaDepth-1500 dataset.

As shown in Table 5, pose estimation accuracy degrades as the number of GNN layers decreases. When using RANSAC for pose estimation, compared to the 5-layer model, the 3-layer variant exhibits AUC drops of 3.2%, 3.0%, and 2.0% at thresholds of 5°, 10°, and 20°, respectively. This indicates that at least five GNN layers are required to maintain sufficient matching accuracy, as fewer layers inadequately refine descriptors, leading to suboptimal matching results.

In terms of inference speed, reducing the network depth by two layers saves approximately 6 ms on GPU and 2.5 seconds on the embedded CPU. Considering both accuracy and efficiency, we conclude that a 5-layer configuration achieves the optimal balance, maintaining competitive matching performance while maximizing inference speed, thereby validating our design choice.

5. Conclusion

In this paper, we address the challenge of deploying deep learning-based image matching algorithms on computationally constrained embedded aerial devices. To improve the efficiency of Transformer-based models in such environments, we proposed FastGlue, a sparse feature matching algorithm that adapts the LightGlue architecture. FastGlue introduces two modifications: replacing the scaled dot-product attention with a ReLU linear attention mechanism, and adopting a shallower graph neural network architecture. These changes are motivated by

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-2-2026-45-2026 | © Author(s) 2026. CC BY 4.0 License.

the computational constraints of embedded CPU platforms and aim to reduce inference time while maintaining competitive accuracy.

Comprehensive experiments on the HPatches and MegaDepth-1500 benchmarks demonstrate that FastGlue achieves matching accuracy comparable to LightGlue, with inference times of 17.05 ms on a GPU and 665.85 ms on an RK3588 embedded CPU—representing a reduction of approximately 15% and 21%, respectively. These results suggest that targeted architectural simplifications can yield meaningful efficiency gains for real-time applications such as UAV visual navigation and target localization, where inference time on embedded hardware is a critical factor.

Several directions remain for future research. First, while our modifications improve inference speed on embedded CPUs, the integration of other efficient architecture designs—such as state-space models or further pruning techniques—could be explored to achieve additional gains. Second, extending the proposed approach to other vision tasks that rely on feature matching, such as image stitching and 3D reconstruction, and validating its performance across more diverse and challenging real-world environments would be valuable next steps.

6. Acknowledgements

This work was funded by Natural Science Foundation of Hubei Province, grant number 2025AFD761.

References

Balntas, V., Lenc, K., Vedaldi, A., Mikolajczyk, K., 2017. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3852–3861.

DeTone, D., Malisiewicz, T., Rabinovich, A., 2017. SuperPoint: Self-Supervised Interest Point Detection and Description. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 337-33712. https://api.semanticscholar.org/CorpusID:4918026.

Dusmanu, M., Rocco, I., Pajdla, T., Pollefeys, M., Sivic, J., Torii, A., Sattler, T., 2019. D2-net: A trainable cnn for joint description and detection of local features. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8084–8093.

Edstedt, J., Athanasiadis, I., Wadenbäck, M., Felsberg, M., 2022. Dkm: Dense kernelized feature matching for geometry estimation.

Edstedt, J., Sun, Q., Bökman, G., Wadenbäck, M., Felsberg, M., 2024. RoMa: Robust Dense Feature Matching. *IEEE Conference on Computer Vision and Pattern Recognition*.

Fischler, M. A., Bolles, R. C., 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6), 381-395.

Gool, T. L. V., 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*.

Katharopoulos, A., Vyas, A., Pappas, N., Fleuret, F., 2020. Transformers are rnns: fast autoregressive transformers with linear attention. *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, JMLR.org.

Larsson, V., contributors, 2020. PoseLib - Minimal Solvers for Camera Pose Estimation.

Li, Z., Snavely, N., 2018. Megadepth: Learning single-view depth prediction from internet photos. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2041–2050.

Lindenberger, P., Sarlin, P.-E., Pollefeys, M., 2023. Lightglue: Local feature matching at light speed. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 17581–17592.

Lowe, D. G., 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), 91-110.

Radenovi, F., Iscen, A., Tolias, G., Avrithis, Y., Chum, O., 2018. Revisiting Oxford and Paris: Large-Scale Image Retrieval Benchmarking. *IEEE*.

Rublee, E., Rabaud, V., Konolige, K., Bradski, G. R., 2011. ORB: an efficient alternative to SIFT or SURF. *IEEE*.

Sarlin, P. E., Detone, D., Malisiewicz, T., Rabinovich, A., 2020. Superglue: Learning feature matching with graph neural networks. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Sun, J., Shen, Z., Wang, Y., Bao, H., Zhou, X., 2021. Loftr: Detector-free local feature matching with transformers.

Wortsman, M., Lee, J., Gilmer, J., Kornblith, S., 2023. Replacing softmax with relu in vision transformers.

Wu, P., Yao, Y., Zhang, W., Wei, D., Wan, Y., Li, Y., Zhang, Y., 2025. Mapglue: Multimodal remote sensing image matching.

Önder Tuzcuoğlu, Köksal, A., Sofu, B., Kalkan, S., Alatan, A. A., 2024. Xoftr: Cross-modal feature matching transformer.