Real-Time SVD Clutter Filtering using Preconditioning

Sebastian Kazmarek Præsius[1], Kees Joost Batenburg[2], Jørgen Arendt Jensen[1]

[1]Department of Health Technology, Technical University of Denmark, Kongens Lyngby, Denmark.
[2]Leiden Institute of Advanced Computer Science, Leiden University, Leiden, the Netherlands.

***Abstract*—Singular value decomposition (SVD) filtering is used in super-resolution and power-Doppler imaging to separate blood flow from the surrounding tissue signals. However, computing the singular vectors is time-consuming, limiting the ability to use the filter for live imaging. It is hypothesized that the vectors that were computed for the previous video frame can be used as a starting point for the calculation of the next frame's vectors, to allow real-time SVD-filtered power-Doppler imaging at a 417 images/s rate. Every image was filtered using the neighboring 401 frames, and to speed up this sliding-window SVD processing, the singular vectors were found by eigendecomposing a temporal covariance matrix. The eigendecomposition iteratively diagonalizes the input matrix, so instead of starting from scratch, the previous frame's vectors were used to precondition (approximately diagonalize) the matrix and allow the next set of vectors to be found in fewer iterations. Batches of 32 frames were computed in parallel to use a graphics processing unit (GPU) more effectively, and the most recent set of vectors was used to precondition the batched eigendecomposition. A GeForce RTX 4090 GPU carried out the processing for in vivo rat kidney data, acquired at a 417 Hz rate with a 10 MHz probe. The resulting rate for the eigendecomposition was 1120 Hz, which was made 60% faster through preconditioning. The imaging rate was 640 Hz after including all processing steps (RF data transfer, beamforming, motion correction, and SVD filtering). Thus, SVD-filtered power-Doppler imaging was performed in real-time using preconditioning, allowing live blood flow imaging at the bedside.**

***Index Terms*—Clutter filtering, Singular value decomposition, SVD, ultrasound localization microscopy, ULM**

I. Introduction

The most popular method to separate blood flow from tissue clutter in super-resolution ultrasound imaging (SRI) is singular value decomposition (SVD)-based spatiotemporal filtering [1]. Such filtering is computationally expensive, but real-time SVD filtered power-Doppler imaging was demonstrated in 2025 [2] using a graphics processing unit (GPU) for the data processing. The processing rate in [2] was 15 872 frames/s (or 31 blocks/s) with one decomposition per block of $N_t = 512$ images, but the most time-consuming calculation was still the decomposition. We propose a preconditioning technique to accelerate this part, which is hypothesized to enable SVD-filtered Doppler imaging using *one decomposition per image* – at a real-time frame rate. This approach avoids block artifacts by filtering every image based on a decomposition of its neighboring $N_t = 401$ frames.

Section II presents the theory of the proposed method. Then, Section III describes the implementation and the measurement setup. The results are shown in Section IV, ending with a short discussion in Section V and concluding remarks in Section VI.

This research was financially supported by the European Research Council's (ERC) Synergy Grant #854796. Corresponding author: Sebastian K. Præsius.

II. Theory

This section describes the theory of the SVD filtering. First, the conventional SVD filter is outlined in Section II-A. Then, Section II-B describes how the filtering can be computed faster through an eigendecomposition, and Section II-C describes the proposed preconditioning to speed up this eigendecomposition.

A. Conventional SVD Filtering

In the conventional SVD filter [3], a sequence of N_t images of size $M_x \times M_z$ are placed into a $M_x M_z \times N_t$ Casorati matrix $\mathbf{Y}$, whose SVD is then computed as the following factorization

$$\mathbf{Y} = \mathbf{U}\mathbf{S}\mathbf{V}^\top = \sum_{i=1}^{N_t} s_{ii}\boldsymbol{u}_i\boldsymbol{v}_i^\top. \tag{1}$$

The result is a diagonal matrix $\mathbf{S}$ with the singular values s_{ii} of $\mathbf{Y}$, and two unitary matrices $\mathbf{U}$ and $\mathbf{V}$, whose columns contain the corresponding singular vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. The filter is then applied by discarding all vectors/values that are not considered to represent blood flow signals, resulting in N_t flow images, as

$$\mathbf{Y}_{\text{flow}} = \mathbf{U}_\mathcal{A}\mathbf{S}_\mathcal{A}\mathbf{V}_\mathcal{A}^\top = \sum_{i\in\mathcal{A}} s_{ii}\boldsymbol{u}_i\boldsymbol{v}_i^\top, \tag{2}$$

where $\mathcal{A}$ is the set of indices considered flow and $\mathbf{U}_\mathcal{A}, \mathbf{S}_\mathcal{A}, \mathbf{V}_\mathcal{A}$ are the submatrices of $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ containing these elements. The next section shows how to accelerate this SVD processing by using $\mathbf{V}$ for the filtration and avoiding the calculation of $\mathbf{U}$.

B. Accelerated SVD Filtering

As described in [3] and [4], the singular vectors $\mathbf{V}$ can be found by eigendecomposing the following $N_t \times N_t$ covariance

$$\mathbf{C} = \mathbf{Y}^\top\mathbf{Y} = \mathbf{V}\mathbf{S}^\top\mathbf{S}\mathbf{V}^\top = \sum_{i=1}^{N_t} s_{ii}^2\boldsymbol{v}_i\boldsymbol{v}_i^\top, \tag{3}$$

which only takes $\mathcal{O}(N_t^3)$ calculations, whereas the SVD in (1) takes $\mathcal{O}(M_x M_z N_t^2)$ calculations assuming $N_t < M_x M_z$ [5]. The $\mathbf{V}$ from (3) can then be used to produce flow images, since $\mathbf{U}_\mathcal{A}\mathbf{S}_\mathcal{A} = \mathbf{Y}\mathbf{V}_\mathcal{A}$ [4], meaning (2) can be computed as

$$\mathbf{Y}_{\text{flow}} = \mathbf{U}_\mathcal{A}\mathbf{S}_\mathcal{A}\,\mathbf{V}_\mathcal{A}^\top = \mathbf{Y}\mathbf{V}_\mathcal{A}\,\mathbf{V}_\mathcal{A}^\top = \mathbf{Y}\mathbf{F}, \tag{4}$$

where $\mathbf{F} = \mathbf{V}_\mathcal{A}\mathbf{V}_\mathcal{A}^\top$ is the $N_t \times N_t$ "SVD filtering matrix". The next section presents the proposed preconditioning technique to speed up the eigendecomposition (3), where $\mathbf{V}$ is computed.

C. Preconditioned SVD Filtering

The decomposition of the symmetric/Hermitian matrix $\mathbf{C}$ in (3) will be computed with the Jacobi eigenvalue algorithm [5], which iteratively updates an *eigenvector estimate* $\hat{\mathbf{V}}$. Each iteration reduces the off-diagonal values $\mathbf{E}$ in $\hat{\mathbf{V}}^\top\mathbf{C}\hat{\mathbf{V}} = \hat{\mathbf{S}}^\top\hat{\mathbf{S}}+\mathbf{E}$; then, as $\|\mathbf{E}\| \to 0$, it converges to the following diagonalization

$$\mathbf{V}^\top\mathbf{C}\mathbf{V} = \mathbf{S}^\top\mathbf{S}, \tag{5}$$

where $\mathbf{V}$ and $\mathbf{S}$ are as in (1). The proposed idea is to apply the singular vectors from the previous frame to precondition $\mathbf{C}$, as

$$\mathbf{C}' = \mathbf{V}_{\text{prev}}^\top\mathbf{C}\mathbf{V}_{\text{prev}}, \tag{6}$$

such that the modified covariance $\mathbf{C}'$ should be approximately diagonal, with relatively low values outside its diagonal, giving a better starting point for the algorithm to reduce the number of iterations needed. Decomposing $\mathbf{C}'$ produces a $\mathbf{V}'$ that satisfies

$$\mathbf{V}'^\top\mathbf{C}'\mathbf{V}' = \mathbf{V}^\top\mathbf{C}\mathbf{V} = \mathbf{S}^\top\mathbf{S}, \tag{7}$$

meaning the solution to the factorization (3) is $\mathbf{V} = \mathbf{V}_{\text{prev}}\mathbf{V}'$, which can be used in the SVD filtering (4). The resulting $\mathbf{Y}_{\text{flow}}$ will be exactly the same as in (2) in theory, but there may be small numerical differences in practice due to round-off errors.

Equation (7) holds because the similarity transformation (6) preserves the eigenvalues of the input $\mathbf{C}$, meaning $\mathbf{C}'$ will have the same eigenvalues as $\mathbf{C}$ [6]. Since $\mathbf{V}_{\text{prev}}$ is unitary, there is no magnification of errors that may be present in the input [5]. Thus, while floating-point arithmetic may add rounding errors, (6) is essentially a best-case scenario that does not reduce the accuracy significantly compared to the method in Section II-B. The proposed method is also numerically stable because it only relies on the $\mathbf{V}_{\text{prev}}$ being unitary (and not that it is error-free), so recursively updating $\mathbf{V}$ will not cause errors to accumulate. However, the hope is that by choosing $\mathbf{V}_{\text{prev}}$ to be the previous frame's vectors, $\mathbf{C}'$ will be *almost diagonal*; if this is the case, then $\|\mathbf{V}' - \mathbf{I}\| \approx 0$, because the identity matrix $\mathbf{I}$ contains the eigenvectors of (and it diagonalizes) any fully diagonal matrix.

III. Method

This section presents the SVD filtering implementation with and without the use of preconditioning in Section III-A. Then, Section III-B describes the RF data and other processing steps. Lastly, Section III-C describes the performance measurements.

A. Implementation

The filtration was mainly implemented with MATLAB code, but the eigendecomposition was carried out in C code to allow it to run efficiently on the GPU (described later in this section). The SVD filtering parameters were $\mathcal{A} = \{32, 33, \ldots, 300\}$ and $N_t = 401$, similar to the parameters in [7] (where $N_t = 400$).

The conventional filtering was implemented as shown in the MATLAB code below, where `Y` is the image data and `A` is $\mathcal{A}$.

```
function Yflow = conventionalSVD(Y, A)
 [U, S, V] = svd(Y, "econ");
 % Now compute the SVD-filtered images.
 Yflow = U(:, A) * S(A, A) * V(:, A)';
end
```

Then, the accelerated filtering was implemented like in [4], by

```
1 function Yflow = acceleratedSVD(Y, A)
2  C = Y'*Y; % Compute covariance matrix.
3  [V, Ssquared] = eig(C, "vector");
4  [~, sidx] = sort(Ssquared, "descend");
5  A = sidx(A); % Match the order in SVD.
6  F = V(:, A) * V(:, A)'; % Make filter.
7  Yflow = Y * F; % Apply the SVD filter.
8 end
```

A reordering is applied on lines 4–5 above to match the SVD, where the components are sorted by descending singular value. As eigenvalues are generally complex, there is no standard way to sort them, but the eigendecomposition of a Hermitian matrix produces only real eigenvalues [6], often returned in ascending order (depending on implementation); here, they were returned in an ascending ordering, so lines 4–5 above could be replaced with "`A=Nt-(A-1);`" to get a descending ordering. Then, the preconditioned SVD filtering was (essentially) implemented as

```
1  function [Yflow, V] = pSVD(Y, A, Vprev)
2   if nargin < 3 % If no Vprev was given.
3       Vprev = eye(size(Y,2)); % Default.
4   end
5   C = Y'*Y; % Compute covariance matrix.
6   C = Vprev'*C*Vprev; % Precondition it.
7   [Vupdate, ~] = eig(C, "vector");
8   A = size(Y,2)-(A-1); % Descend. order.
9   V = Vprev * Vupdate; % Update vectors.
10  F = V(:, A) * V(:, A)'; % Make filter.
11  Yflow = Y * F; % Apply the SVD filter.
12 end
```

where the vectors `V` are returned as a second output argument, so they can be passed in the next call to the function as `Vprev`.

As $\mathbf{C}$ was relatively small, only 401×401, multiple decompositions were computed in parallel to better utilize the GPU. Since the MATLAB function `pageeig` does not support GPU inputs, a MEX file (external C library) was needed to manually call the eigendecomposition function for a batch of Hermitian matrices "`heevjBatched`"[1] from the cuSOLVER library [8]. This MEX file returned the filters $\mathbf{F}_i$ given B covariances $\mathbf{C}_i$, meaning lines 6–10 above were implemented in C code. To run the calculation in the background during other processing steps (e.g., beamforming and motion correction), GPU operations in the MEX file were issued into a dedicated CUDA stream (task queue), created using the `cudaStreamNonBlocking` flag. However, the solver `heevjBatched` was not asynchronous, so a dedicated CPU thread was spawned in the MEX file to call it, making the calculation of filters fully asynchronous. Thus, B filters were computed in the background, and the result was retrieved when submitting the next set of covariance matrices. The latest singular vectors $\mathbf{V}_{\text{prev}}$ were applied to all inputs $\mathbf{C}_i$.

To compute an approximate decomposition, a tolerance can be set to stop iteratively updating the estimated eigenvectors $\hat{\mathbf{V}}$

[1]The naming convention is: *he* = Hermitian, *ev* = eigenvalue, *j* = Jacobi.

(see Section II-C) after the error $\mathbf{E}$ satisfies $\|\mathbf{E}\| \leq \varepsilon \|\mathbf{C}\|$. The tolerance was set to $\varepsilon = 2^{-18} \approx 3.81 \cdot 10^{-6}$ (32 times machine epsilon) to accelerate the processing, and it was investigated in the results whether this reduced the image quality (see Fig. 1).

Each image was filtered using the adjacent $N_t = 401$ frames and their covariances (i.e., a sliding window, like in [7]). This was implemented by calculating the covariances between every new frame and the previous $N_t - 1$ frames, like in [9, Sec 3.4]. Thus, for every frame, there was a matrix with the covariances of its neighboring N_t frames, which was converted into a filter, and the output was this one frame, after filtering, computed as

$$\boldsymbol{y}_{\text{flow}} = \mathbf{Y}\boldsymbol{f}, \tag{8}$$

where $\boldsymbol{f}$ was the central column of the filter $\mathbf{F}$. The covariance and the output (8) were also computed for batches of B frames. Thus, in the final implementation, each output (8) contained B blood flow images that were all filtered *as if* they had been the central frame in (2), where $\mathbf{Y}$ was the neighboring N_t frames. The batch size was set to $B = 32$ in the final implementation.

B. Data Acquisition and Additional Processing Steps

The RF data from [7] was used, which was acquired in vivo from a Sprague Dawley rat kidney with a 10 MHz GE L8-18iD linear array probe and a Verasonics Vantage 256 scanner. The images were captured at a 417 Hz frame rate using a synthetic aperture sequence containing twelve defocused emissions [7]. The processing included beamforming, motion correction, and SVD filtering. First, the GPU-optimized beamformer from [10] was used to produce $M_x \times M_z = 354 \times 390$ size images. The motion between frames was estimated with the autocorrelation motion estimator in [7]. To keep the tissue stationary and allow it to be separated from slow blood flow, the image from each emission was motion corrected using a modified beamformer, resulting in motion-corrected images, ready for SVD filtering. Finally, the squared envelope (i.e., absolute value) of the SVD-filtered images was accumulated to create the power-Doppler.

Like in Section III-A, every processing step was carried out in parallel for B frames at a time to utilize the GPU effectively. This is because the device could contain up to 196 608 threads (parallel workers), but $354 \cdot 390 = 138\,060$ pixels would result in only 138 060 threads when each thread computes one pixel, leading to an underutilization of the computational hardware. However, the full details of the batched pipeline are outside the scope of this work, as it is not important to the preconditioning.

C. Performance Measurement

The processing was measured in MATLAB R2024b with an NVIDIA GeForce RTX 4090 graphics card. This was done by wrapping the processing code in a function and then measuring its execution time using the MATLAB function `gputimeit`. The processing computed the power-Doppler image from the first $N = 5000$ frames (12 seconds) of data, and its processing rate was computed as the average rate R in

$$R = \frac{1}{N_{\text{reps}}} \sum_{i=1}^{N_{\text{reps}}} R_i, \quad \text{with } R_i = \frac{N}{T_i}, \tag{9}$$

where T_i was the measured execution times after repeating the experiment $N_{\text{reps}} = 15$ times. Real-time imaging is achieved if this rate R exceeds the acquisition rate of 417 Hz. The rate of the eigendecomposition was also measured in isolation, and in this case, only a single batch of $N = B$ inputs was measured.

IV. Results

This section presents the results for the preconditioned SVD processing. The power-Doppler images resulting from the conventional filtering versus the proposed preconditioned filtering can be compared in Fig. 1, where it can be seen that the images are virtually indistinguishable. Thus, the image quality was not worsened (or improved) by the proposed SVD implementation.

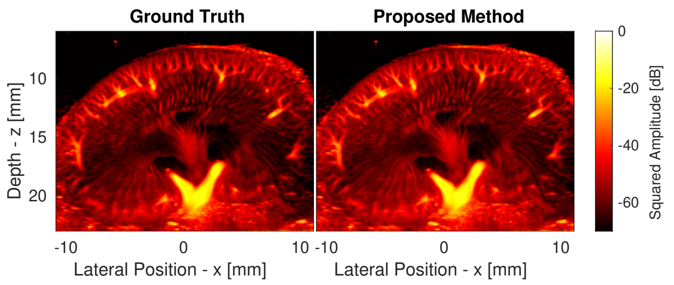

Figure 1. The image from the conventional SVD filtering in double precision (ground truth), and the image from the proposed preconditioned SVD filtering.

The processing rate was 640 Hz, including all processing steps, demonstrating real-time SVD-filtered power-Doppler imaging. Without preconditioning, processing rate was 417 Hz when all processing steps were included. The speed of the SVD filtering step was 677 Hz (or 572 Hz without using preconditioning).

The preconditioning relies on the change in the singular vectors $\mathbf{V}$ being limited between frames, allowing their difference $\mathbf{V}' = \mathbf{V}_{\text{prev}}^{-1}\mathbf{V}$ to be computed quickly through (7). One can see that this was the case in Fig. 2, which shows an example of $\mathbf{V}'$, where the right image was made after the following reordering:

```
Vprev = Vprev([2:end,1], :); % Circshift.
```

to account for the fact that most entries in the covariance were identical between neighboring frames (but shifted by one slot).

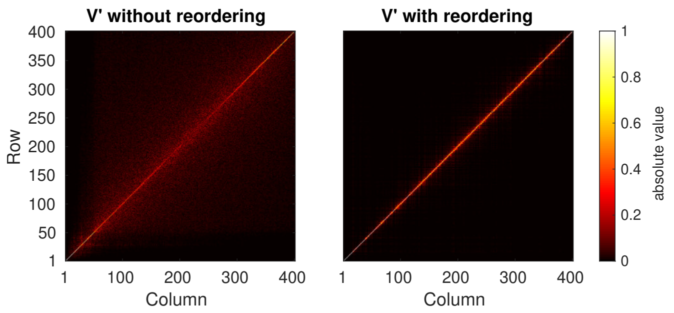

Figure 2. The difference in the singular vectors between two arbitrary frames. The difference was so small with reordering, $\mathbf{V}'$ was nearly an identity matrix.

The reordering above increased the proximity of $\mathbf{V}_{\text{prev}}$ to $\mathbf{V}$, improving the processing speed, as seen in Fig. 3, which shows the eigendecomposition rate for $401 \times 401 \times B$ covariances $\mathbf{C}_i$. Reordering was not used in the final implementation, however, because it has yet to be implemented in the MEX file. The rate peaked at $B = 27$ in Fig. 2, but the optimal value was lower for larger inputs. For example, the peak rate for a 512×512 matrix was 1011 Hz at $B = 13$, and more results are shown in Table I. The rate for a $512 \times 512 \times 1$ input was 266 Hz using reordered preconditioning (with `heevj`), but for `heevd`, it was 275 Hz.

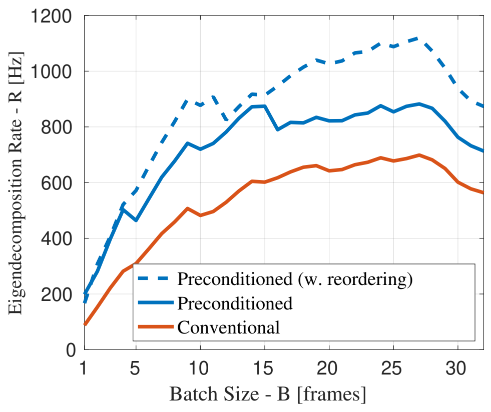

Figure 3. The eigendecomposition processing rate for varying batch sizes.

Table I
EIGENDECOMPOSITION PROCESSING RATE (IN MATRICES/S).

N_t	Baseline	Proposed Methods		Speed-up
	Conventional	**Preconditioned**	**w. Reordering**	
256	2811	3853	**4933**	76 %
401	699	882	**1120**	60 %
512	403	630	**1011**	151 %
1024	52	89	**118**	127 %

V. Discussion

The iterative Jacobi algorithm "`heevj`" was accelerated by preconditioning. It supports batch processing to enable faster processing of small matrices, and one can adjust its tolerance ε for further acceleration [8]. The divide-and-conquer algorithm "`heevd`" was used in [2], which may be faster for large inputs, but `heevj` was 3.7 times faster when $N_t = 512$ due to the use of batch processing. (Otherwise, they achieved similar speeds.) Faster processing allows a greater number of frames to be used in the SVD filter, which potentially improves its sensitivity to slow flow (e.g., in small blood vessels), as slow flow resembles still tissue on short time-scales, so longer observation times are necessary to separate it. The proposed method has applications in ultrasound localization microscopy (ULM), which still does not have a real-time implementation, but SRI/ULM processing only requires adding a few processing steps after the filtration.

The time for image display was not measured. Transferring images to the PC screen is computationally trivial, but a real-time display requires more custom code because the MATLAB function `imagesc` was too slow for real-time use. This would be tedious to implement, and it would not be novel (e.g., image display was developed in [2]), so it was left out of the scope of this work. It was shown that all processing steps until this point (RF data transfer to the GPU, beamforming, motion correction, SVD filtering, image formation) ran at a real-time frame rate.

VI. Conclusion

The eigendecomposition was the most time-consuming part of the processing, taking up 60% of the total processing time. This was the motivation behind the proposed preconditioning, which increased the speed of this part by over 60%, enabling real-time SVD-filtered Doppler imaging at a 640 frames/s rate. The proposed method removes one of the major barriers to the clinical adoption of SVD filtering by enabling the visualization of blood flow at high frame rates in real time in the live setting.

References

[1] S. Dencks *et al.*, “Super-resolution ultrasound: From data acquisition and motion correction to localization, tracking, and evaluation,” *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 72, no. 4, pp. 408–426, 2025.

[2] B. Pialot *et al.*, “Computationally efficient SVD filtering for ultrasound flow imaging and real-time application to ultrafast Doppler,” *IEEE Trans. Biomed. Eng.*, vol. 72, no. 3, 2025.

[3] C. Demené *et al.*, “Spatiotemporal clutter filtering of ultrafast ultrasound data highly increases Doppler and fUltrasound sensitivity,” *IEEE Trans. Med. Imag.*, vol. 34, no. 11, pp. 2271–2285, 2015.

[4] B. Pialot, L. Augeul, L. Petrusca, and F. Varray, “A simplified and accelerated implementation of SVD for filtering ultrafast power Doppler images,” *Ultrasonics*, vol. 134, p. 107 099, 2023.

[5] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 4th ed. The Johns Hopkins University Press, 2013.

[6] G. Strang, *Introduction to linear algebra*, 4th ed. Wellesley Cambridge Press, 2009.

[7] J. A. Jensen *et al.*, “Super-resolution ultrasound imaging using the erythrocytes—Part I: Density images,” *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 71, no. 8, pp. 925–944, 2024.

[8] NVIDIA, “cuSOLVER API reference (version 12.8),” NVIDIA Corporation, Tech. Rep., 2025, Accessed June 2025. [Online]. Available: https://docs.nvidia.com/cuda/archive/12.8.0/cusolver/index.html#cusolverdn-t-syevjbatched.

[9] S. D. Pendergrass, J. N. Kutz, and S. L. Brunton, *Streaming GPU singular value and dynamic mode decompositions*, 2016. [Online]. Available: https://arxiv.org/abs/1612.07875.

[10] S. K. Præsius, L. T. Jørgensen, and J. A. Jensen, “Real-time full-volume row-column imaging,” *IEEE Trans. Ultrason. Ferroelec. Freq. Contr.*, vol. 72, no. 1, pp. 109–126, 2025.