How to Design Fast GFTs

Speeding up the computation of the Graph Fourier Transform: symmetries, decompositions, approximations, and applications

Antonio Ortega

University of Southern California

Joint work with: **Samuel Fernández Menduiña, Keng-Shi Lu, Darukeesan Pakiyarajah, Eduardo Pavez**

June 8, 2026

Motivation

Graph Fourier Transform (GFT)

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top, \qquad \hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f}$$

- Generalizes classical transforms to irregular domains
- Unlike the DFT/DCT: **no single fast algorithm** that works for **all graphs**.

Problem

- Different graphs – different structures: path, ring, tree, sensor network, mesh, social graph...
- Direct GFT (given eigendecomposition of $\mathbf{L}$): $\mathcal{O}(N^2)$ — prohibitive for large N
- Applications in image/video coding and graph ML demand scalable transforms

Core Challenge

How do we exploit **graph-specific properties to design fast GFT algorithms** — analogous to how the FFT exploits the circulant structure of the DFT?

Background

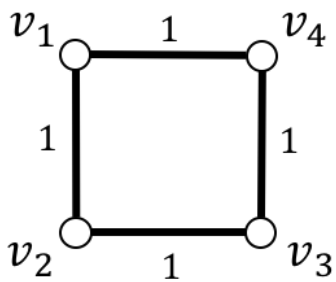
The Path / Circulant Graph

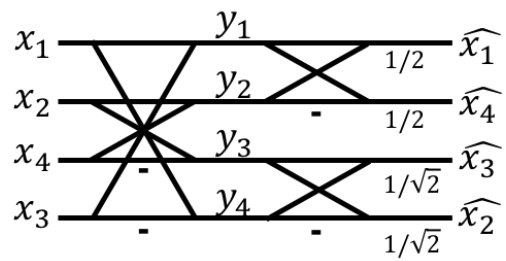
The DFT is the GFT of the directed circulant graph (DCT is the GFT of the path graph). The graph Laplacian $\mathbf{L}$ is a *circulant* matrix:

$$\mathbf{L}_{\text{circ}} = \text{circ}(2, -1, 0, \ldots, 0, -1) \;\Rightarrow\; \mathbf{U} = \mathbf{F}_N$$

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n]\, e^{-j2\pi nk/N}$$

Key insight: The FFT is fast because the circulant graph has perfect symmetry — it can be recursively decomposed into two half-size problems. $\mathcal{O}(N \log N)$ **GFT via FFT**.





Goal in this talk: Identify and exploit graph-specific structure to achieve faster (exact or approximate) GFT computation (less than $\mathcal{O}(N^2)$) for broader graph families.

Outline

Four Approaches to Fast GFTs

01

Frequency Domain

Direct approximations in the spectral domain using Givens rotations.

02

Vertex Domain

Spectral sparsification of the graph — replace $\mathcal{G}$ with a sparser $\tilde{\mathcal{G}}$.

03

Graph Symmetries

Use graph symmetries to decompose the GFT into smaller subproblems.

04

Graph Clustering

Leverage graph decompositions via low-rank updates — partition the graph into clusters.

Frequency Domain

Approximate Fast GFT via Multi-Layer Sparse Factorization [Le Magoarou et al, TSIP 2017]

Idea: Approximate the GFT matrix $\mathbf{U}$ as a product of J sparse, orthogonal Givens rotation matrices — avoiding full diagonalization.

Approximate Diagonalization

$$\mathbf{L} \approx \mathbf{S}_1 \cdots \mathbf{S}_J \hat{\mathbf{\Lambda}} \mathbf{S}_J^T \cdots \mathbf{S}_1^T$$

$$\hat{\mathbf{U}} = \mathbf{S}_1 \cdots \mathbf{S}_J, \quad \mathbf{S}_j \in \mathcal{S} \text{ (sparse \& orthogonal)}$$

Optimization Objective (Frobenius norm)

$$\min_{\hat{\mathbf{\Lambda}}, \mathbf{S}_1, \ldots, \mathbf{S}_J} \left\| \mathbf{L} - \mathbf{S}_1 \cdots \mathbf{S}_J \hat{\mathbf{\Lambda}} \mathbf{S}_J^\top \cdots \mathbf{S}_1^\top \right\|_F^2$$

- Each $\mathbf{S}_j$ is a Givens rotation — acts on a 2D subspace
- Factors are sparse and orthogonal $\Rightarrow$ easily invertible
- Overall cost $\mathcal{O}(J)$
- Approach: modified Jacobi eigenvalue algorithm

Givens Rotation

$\mathbf{G}(i, j, \theta)$ acts on the 2D subspace spanned by $\mathbf{e}_i, \mathbf{e}_j$, rotating by θ:

$$\mathbf{G}(i,j,\theta) = \mathbf{I} + (\cos\theta - 1)(\mathbf{e}_i\mathbf{e}_i^\top + \mathbf{e}_j\mathbf{e}_j^\top) + \sin\theta\,(\mathbf{e}_j\mathbf{e}_i^\top - \mathbf{e}_i\mathbf{e}_j^\top)$$

Sparse, orthogonal

Algorithm Steps

1. Initialize $\hat{\mathbf{U}} = \mathbf{I}$, residual $\mathbf{R} = \mathbf{L}$
2. Find $(i^*, j^*) = \arg\max_{i \neq j} |\mathbf{R}_{ij}|$
3. Compute $\theta^* = \frac{1}{2}\arctan\left(\frac{2\mathbf{R}_{i^*j^*}}{\mathbf{R}_{i^*i^*} - \mathbf{R}_{j^*j^*}}\right)$
4. Update: $\mathbf{R} \leftarrow \mathbf{G}^\top\mathbf{R}\mathbf{G}$, $\hat{\mathbf{U}} \leftarrow \hat{\mathbf{U}}\mathbf{G}$
5. Repeat for K steps; accumulate $\hat{\mathbf{U}} = \mathbf{G}_1\mathbf{G}_2\cdots\mathbf{G}_K$

Greedy Optimality

Each step maximally reduces the off-diagonal Frobenius norm:

$$\left\|\mathbf{R}^{(k+1)}\right\|_F^2 = \left\|\mathbf{R}^{(k)}\right\|_F^2 - 2\,(\mathbf{R}^{(k)}_{i^*j^*})^2$$

Convergence guaranteed: $\left\|\mathbf{R}^{(K)}_{\text{off}}\right\|_F \to 0$ as $K \to \infty$.

Stopping at $K = \mathcal{O}(N \log N)$ gives a fast GFT, but approximation quality depends on the graph

Contrast with classical Jacobi: classical sweeps all pairs; this greedy variant selects the most impactful pair at each step — same convergence, better sparsity.

Relative Complexity Gain (RCG)

$$\mathrm{RCG} \triangleq \frac{\|\mathbf{U}\|_0}{\sum_{j=1}^{J} \|\mathbf{S}_j\|_0}$$

$\mathrm{RCG} = 10 \Rightarrow 10\times$ fewer multiplications & $10\times$ less storage.

Error–Speed Tradeoff As K increases, approximation error ϵ decreases while RCG also decreases

Approximation Results Random sensor networks

	$n = 64$	$n = 128$	$n = 256$	$n = 512$	$n = 1024$	$n = 2048$	$n = 4096$	$n = 8192$
RCG	1.33	2.29	4.00	7.11	12.80	23.27	42.67	78.77
Time gain	0.04	0.05	0.11	0.26	1.56	3.88	7.57	27.16
$\mathrm{err}_d(\hat{\mathbf{U}}_{\mathrm{Givens}//})$	0.057	0.062	0.055	0.051	0.048	0.051	0.049	0.048

Next Practical use case for video coding

Motivation & Problem Setting

Secondary transforms decorrelate residual coefficients of a separable primary transform (e.g. DCT) in video codecs. State-of-the-art LFNST truncates high-frequency coefficients to reduce cost — limiting coding gains.

Core Idea: SOT $\rightarrow$ Givens Factorization

Learn a *sparse orthonormal transform* (SOT) with an RD-inspired cost, then constrain it to be a product of J Givens rotations.

Optimization Objective

$$\min_{S_J,\,\{\hat{y}_i\}} \sum_{i=1}^{m} \|\hat{x}_i - S_J \hat{y}_i\|_2^2 + \mu \|\hat{y}_i\|_0 \quad \text{s.t. } S_J = \prod_{j=1}^{J} \mathbf{G}(m_j, n_j, \theta_j)$$

$\mu = (Q_s/2)^2$ ties sparsity to quantization step Q_s — explicit RD-aware design.

Pakiyarajah, Fernández-Menduiña, Pavez, Ortega, Mukherjee — "FaSST: Fast Sparsifying Secondary Transform," *ICIP 2026*. [arXiv:2605.15086]

Frequency Domain

FaSST: Fast Sparsifying Secondary Transform — Comparison with KLT

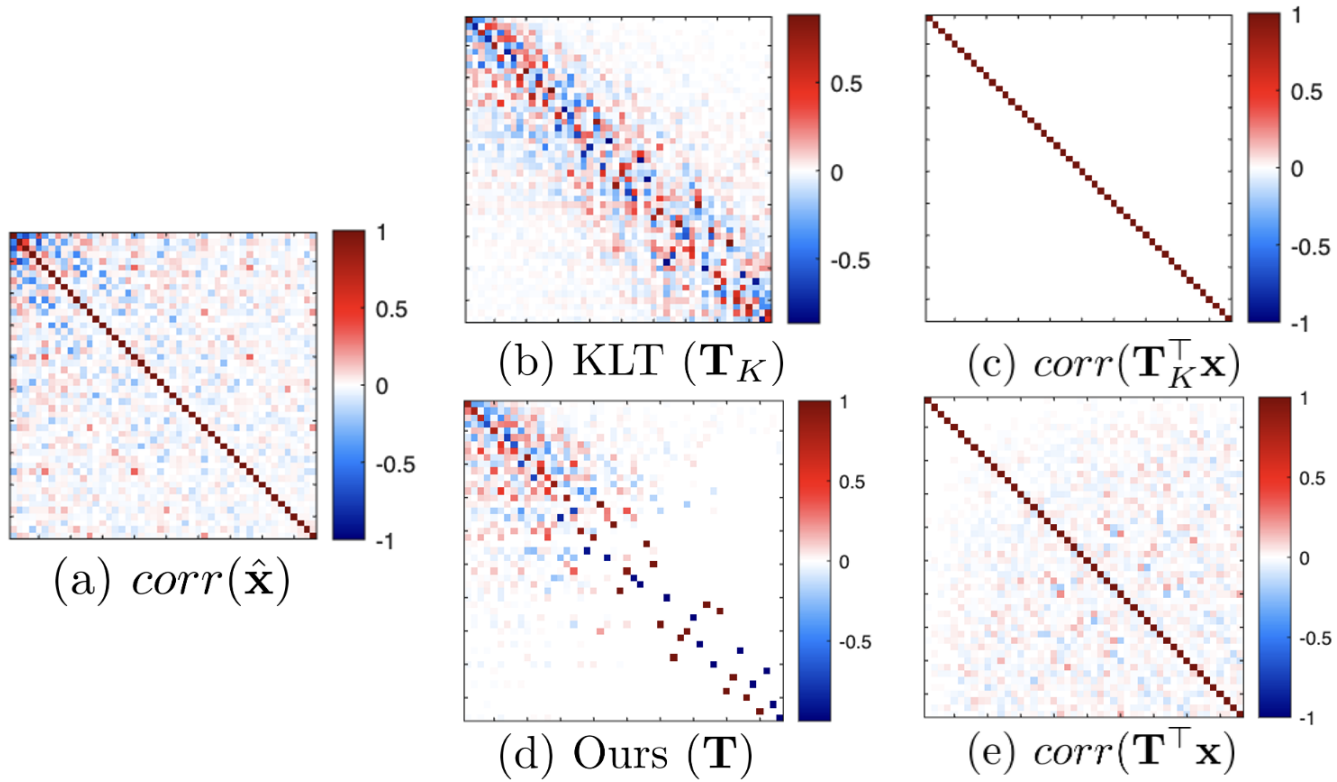


Figure: Correlation matrices and transform kernels for AV2 intra-prediction residuals (D_{135} mode): (a) correlation of 48 (out of 16^2) sorted DCT coefficients, (b) KLT kernel (48×48), (c) correlation after KLT, (d) proposed FaSST (48×48), and (e) correlation after the proposed transform.

Frequency Domain

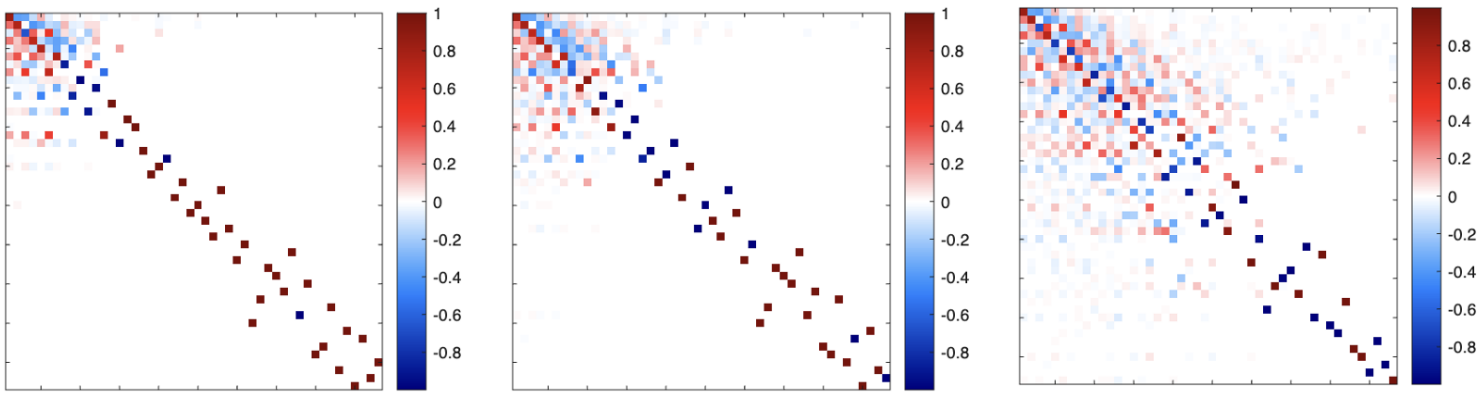


Figure: FaSST kernels designed using $\tau = 10^{-6}$ and different numbers of Givens rotations: (a) $J = 64$, (b) $J = 128$, and (c) $J = 384$.

Frequency Domain

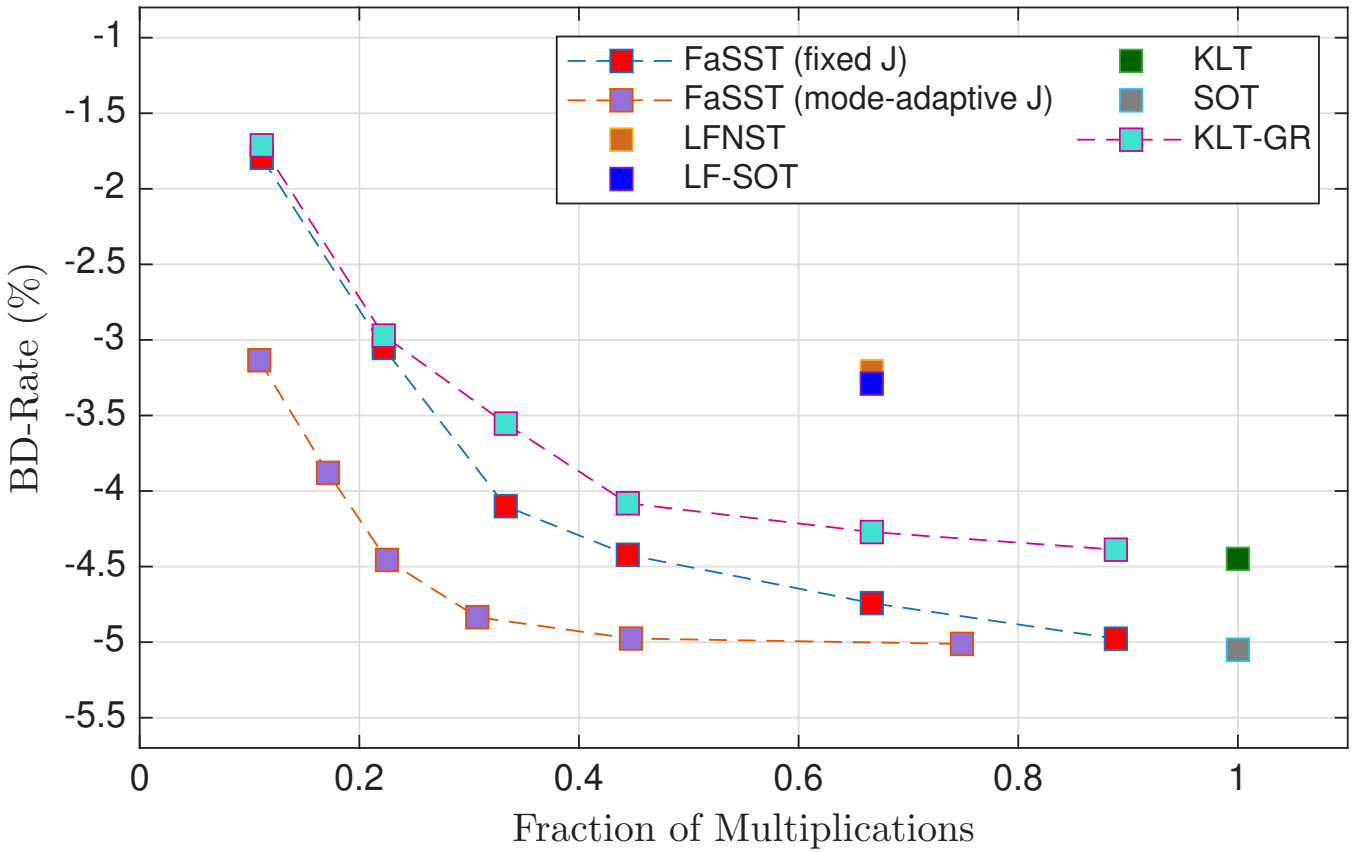


Figure: Average BD-rate savings (more negative is better) versus fraction of multiplications relative to KLT for FaSST (fixed/mode-adaptive J), LFNST LF-SOT, KLT, SOT , and KLT-GR .

.

Instead of approximating $\mathbf{U}$ directly, **sparsify the graph** $\mathcal{G}$ to obtain $\tilde{\mathcal{G}}$ with fewer edges

- GFT may be cheaper to compute (more on this later)

Spectral Sparsification

Find $\tilde{\mathbf{L}}$ (sparse) such that:

$$(1-\epsilon)\,\mathbf{L} \preceq \tilde{\mathbf{L}} \preceq (1+\epsilon)\,\mathbf{L}$$

- $\tilde{\mathbf{L}}$ has fewer edges (vs $\mathcal{O}(N^2)$ for dense $\mathbf{L}$)
- Preserves spectral properties: eigenvalues of $\tilde{\mathbf{L}}$ approximate those of $\mathbf{L}$
- Controlled approximation error in the spectral domain

Vertex Domain

Key Insight

A sparse graph $\tilde{\mathcal{G}}$ with $\mathcal{O}(N \log N)$ edges can spectrally approximate a dense graph with $\mathcal{O}(N^2)$ edges — preserving all eigenvalue information up to $(1 \pm \epsilon)$.

Sparsification Procedure

1. Compute effective resistance $R_e(u, v)$ for each edge
2. Sample each edge with probability $p_{uv} \propto w_{uv} \cdot R_e(u, v)$
3. Rescale sampled edge weights: $\tilde{w}_{uv} = w_{uv}/p_{uv}$ (unbiased estimator)
4. Result: $\tilde{\mathcal{G}}$ with $\mathcal{O}(N \log N/\epsilon^2)$ edges

Spectral Guarantee

$$(1 - \epsilon)\,\mathbf{L} \preceq \tilde{\mathbf{L}} \preceq (1 + \epsilon)\,\mathbf{L}$$

$\Leftrightarrow$ for all signals $f \in \mathbb{R}^N$:

$$(1 - \epsilon)\,\mathbf{f}^\top \mathbf{L}\mathbf{f} \leq \mathbf{f}^\top \tilde{\mathbf{L}}\mathbf{f} \leq (1 + \epsilon)\,\mathbf{f}^\top \mathbf{L}\mathbf{f}$$

Eigenvalues of $\tilde{\mathbf{L}}$ lie in $[(1 - \epsilon)\lambda_k, (1 + \epsilon)\lambda_k]$.

Effective Resistance

The effective resistance between nodes u and v measures connectivity:

$$R_e(u, v) = (\mathbf{e}_u - \mathbf{e}_v)^\top \mathbf{L}^\dagger (\mathbf{e}_u - \mathbf{e}_v)$$

where $\mathbf{L}^\dagger$ is the Moore–Penrose pseudoinverse of $\mathbf{L}$.

Sampling Probability

$$p_{uv} = \min\left(1,\ c \cdot \frac{\log N}{\epsilon^2} \cdot w_{uv} \cdot R_e(u, v)\right)$$

Edges with high $w_{uv} R_e(u, v)$ are more important spectrally.

Intuition: Assume two connected nodes ($w_{uv} \neq 0$)

- No other connecting paths between u and v ($R_e(u, v) = 1/w_{uv}$) $\longrightarrow$ Sampling probability ≈ 1
- Multiple parallel paths $\longrightarrow$ $R_e(u, v) < 1/w_{uv}$

Spielman–Srivastava (2011)

With $c \cdot N \log N/\epsilon^2$ sampled edges, the sparsifier $\tilde{L}$ satisfies $(1-\epsilon)L \preceq \tilde{L} \preceq (1+\epsilon)L$ with high probability.

Properties of the Sparsifier

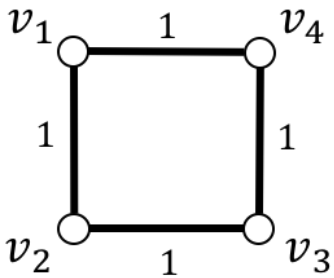
- All eigenvalues of $\tilde{\mathbf{L}}$ approximate those of $\mathbf{L}$ within factor $(1 \pm \epsilon)$
- Eigenvectors of $\tilde{\mathbf{L}}$ are close to those of $\mathbf{L}$ (Davis–Kahan theorem)
- Graph signal smoothness preserved: $\mathbf{f}^\top \tilde{\mathbf{L}}\mathbf{f} \approx \mathbf{f}^\top \mathbf{L}\mathbf{f}$

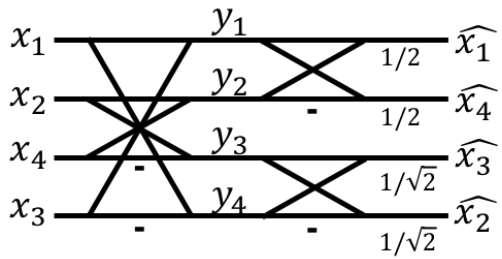
Connection to GFT: Simplifying $\mathbf{L}$ can make it practical to use filters $h(\mathbf{L})$ for frequency analysis, instead of using the GFT.

Additionally: Can lead to graphs with more favorable properties for fast GFT (different GFT, but with spectral approximation guarantees).

Graph Symmetries

Recall: Role of symmetry in the FFT





Question: Can this be generalized to other types of symmetries that may appear in graphs?

Core Contribution: For graphs with bipartition or topological symmetry (node-pairing involutions), the GFT can be factored into butterfly stages of Haar units — yielding an **exact** fast GFT with no approximation error.

K.-S. Lu & A. Ortega, "Fast Graph Fourier Transforms Based on Graph Symmetry and Bipartition," *IEEE Trans. Signal Inf. Process. Netw.*, 2019. [arXiv:1907.07875]

Graph Symmetries

Definition: Involution and ϕ-symmetry
A permutation $\phi : \mathcal{V} \to \mathcal{V}$ is an *involution* if $\phi(\phi(i)) = i$ for all $i \in \mathcal{V}$.
Graph $\mathcal{G}$ is ϕ-symmetric if

$$w_{i,j} = w_{\phi(i),\phi(j)} \quad \forall\, i, j \in \mathcal{V}$$

Key Insight

A graph $\mathcal{G}$ is ϕ-symmetric under involution ϕ if $w_{i,j} = w_{\phi(i),\phi(j)}$ for all $i, j \in \mathcal{V}$. This symmetry enables a divide-and-conquer decomposition:

$$\mathbf{L} \xrightarrow{\mathbf{B}_\phi} \operatorname{diag}(\mathbf{L}^+, \mathbf{L}^-)$$

two independent sub-GFTs of roughly half the size.

Node Partition Given involution ϕ, partition nodes into three sets:

$$\mathcal{V}_Z = \{i \in \mathcal{V} : \phi(i) = i\} \quad \text{(symmetry axis)}$$
$$\mathcal{V}_X, \mathcal{V}_Y \quad \text{paired nodes, } |\mathcal{V}_X| = |\mathcal{V}_Y| = p_\phi$$

Haar Unit (Butterfly) Only 1 addition + 1 subtraction (no multiplications after absorbing $1/\sqrt{2}$).

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Examples of ϕ-Symmetric Graphs

Graph	Symmetry
Line P_N	$\phi = (N, N-1, \ldots, 1)$ — reflection around center. GFT = DCT.
Cycle C_N	$\phi = (N, N-1, \ldots, 2, 1)$ — reflection. GFT = DFT. At least one butterfly stage for any N.
Skeletal	Human body: left/right arm symmetry $\Rightarrow$ paired branches $\Rightarrow$ butterfly stage for action analysis.

Key Distinction from Approach 1

Symmetry-based fast GFTs are *exact* — no approximation error. Speedup comes from graph topology, not optimization.

Butterfly Matrix $\mathbf{B}_\phi$

For a ϕ-symmetric graph with partition $\mathcal{V}_X, \mathcal{V}_Y, \mathcal{V}_Z$:

$$(\mathbf{B}_\phi)_{i,j} = \begin{cases} 1/\sqrt{2}, & i = j \in \mathcal{V}_X \\ -1/\sqrt{2}, & i = j \in \mathcal{V}_Y \\ 1, & i = j \in \mathcal{V}_Z \\ 1/\sqrt{2}, & i \in \mathcal{V}_X,\ j = \phi(i) \in \mathcal{V}_Y \\ 1/\sqrt{2}, & i \in \mathcal{V}_Y,\ j = \phi(i) \in \mathcal{V}_X \\ 0, & \text{otherwise} \end{cases}$$

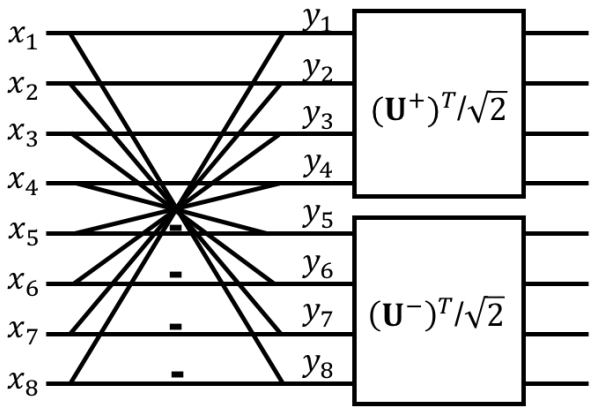
Block Diagonalization

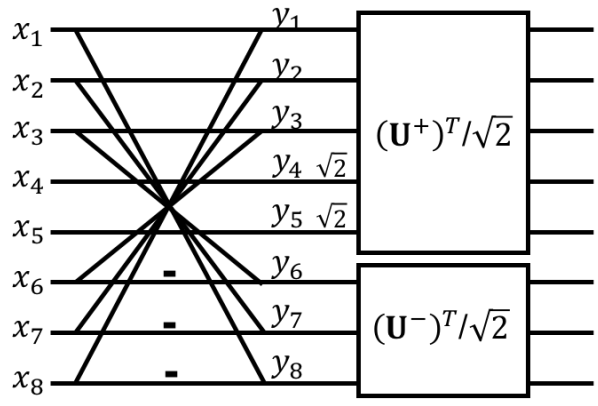
$$\mathbf{L}_\phi := \mathbf{B}_\phi^T \mathbf{L}\, \mathbf{B}_\phi = \mathrm{diag}(\mathbf{L}^+, \mathbf{L}^-)$$

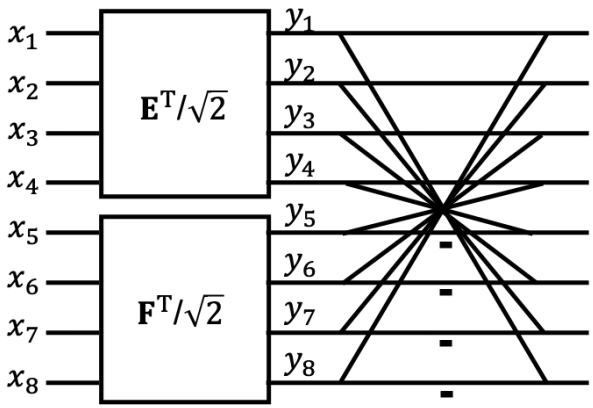
- $\mathbf{L}^+ \in \mathbb{R}^{(n-p_\phi)\times(n-p_\phi)}$ — "sum" (low-pass) sub-graph $\mathcal{G}^+$
- $\mathbf{L}^- \in \mathbb{R}^{p_\phi \times p_\phi}$ — "difference" (high-pass) sub-graph $\mathcal{G}^-$

Graph Symmetries

Left and Right butterfly stages







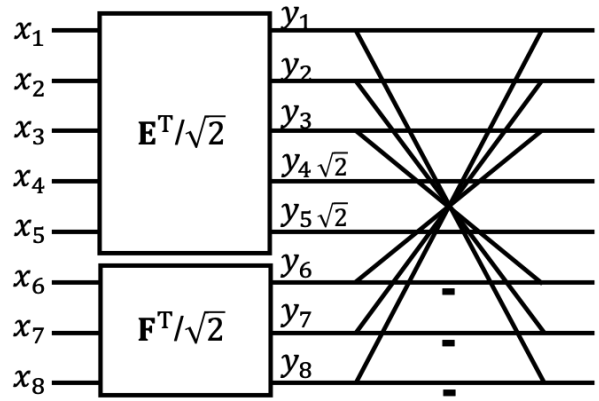


Figure: Examples of fast algorithms using butterfly stages with $n = 8$. Top: Left butterfly stages. Bottom: Right butterfly stages (bipartite graphs).

Graph Symmetries

Divide-and-Conquer Decomposition via Butterfly Stage $\mathbf{B}_\phi$

Fast GFT Algorithm

1. Find involution ϕ such that $\mathcal{G}$ is ϕ-symmetric
2. Partition nodes: $\mathcal{V}_Z = \{i : \phi(i) = i\}$, then $\mathcal{V}_X, \mathcal{V}_Y$
3. Apply butterfly stage $\mathbf{B}_\phi$: compute $\mathbf{L}^+$ and $\mathbf{L}^-$ from $\mathbf{L}$
4. Recurse: check if $\mathcal{G}^+$ and $\mathcal{G}^-$ are themselves ϕ'-symmetric
5. GFT: $\mathbf{U} = \mathbf{B}_\phi \cdot \mathrm{diag}(\mathbf{U}^+, \mathbf{U}^-)$

Sub-graphs

$$\mathbf{L}^+ = \mathbf{L}_{XX} + \mathbf{L}_{XY}\mathbf{J} \quad (\text{“sum” signal})$$
$$\mathbf{L}^- = \mathbf{L}_{YY} - \mathbf{J}\mathbf{L}_{XY} \quad (\text{“difference” signal})$$

Complexity

One butterfly stage: $p_\phi^2 + (n - p_\phi)^2$ multiplications — minimized to $n^2/2$ when $p_\phi = n/2$ ($2\times$ speedup). Recursive stages $\Rightarrow \mathcal{O}(N \log N)$ for fully symmetric graphs.

Speed Performance (Lu & O. 2019)

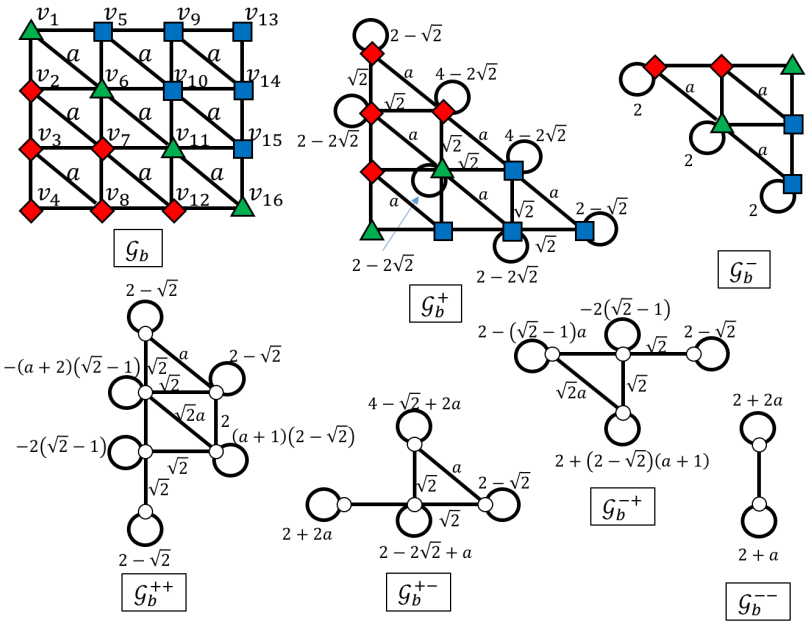
Table: Speed performance of proposed fast GFTs. The baseline for the runtime reduction rates is the matrix GFT implementation.

Topology	n	Number of Operations		Runtime
		Matrix ($\pm/*$)	Fast ($\pm/*$)	Reduction
Cycle	12	132/144	44/30	52.7%
	80	6320/6400	1224/1078	79.7%
6-conn. grid	16	240/256	80/80	53.7%
	64	4032/4096	1104/1072	68.5%
Z-shaped grid	16	240/256	128/112	41.5%
	64	4032/4096	2048/2048	45.0%
Skeleton	15	210/225	96/102	45.5%
	25	600/625	272/282	47.5%

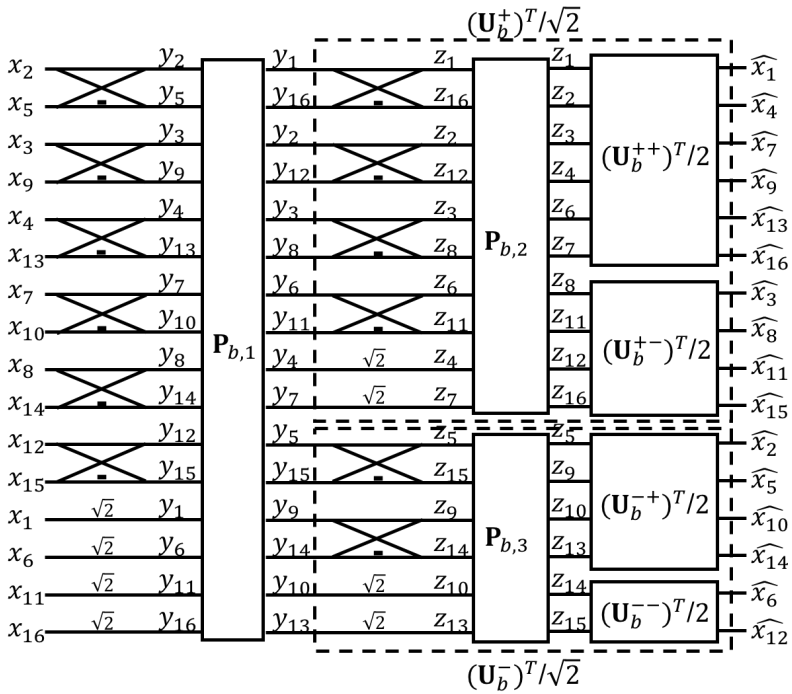
Graph Symmetries

BD Grid Graph

Grid graph



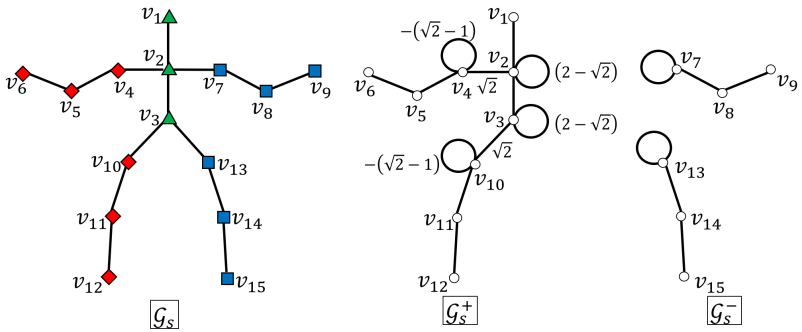
Transform for Grid Graph



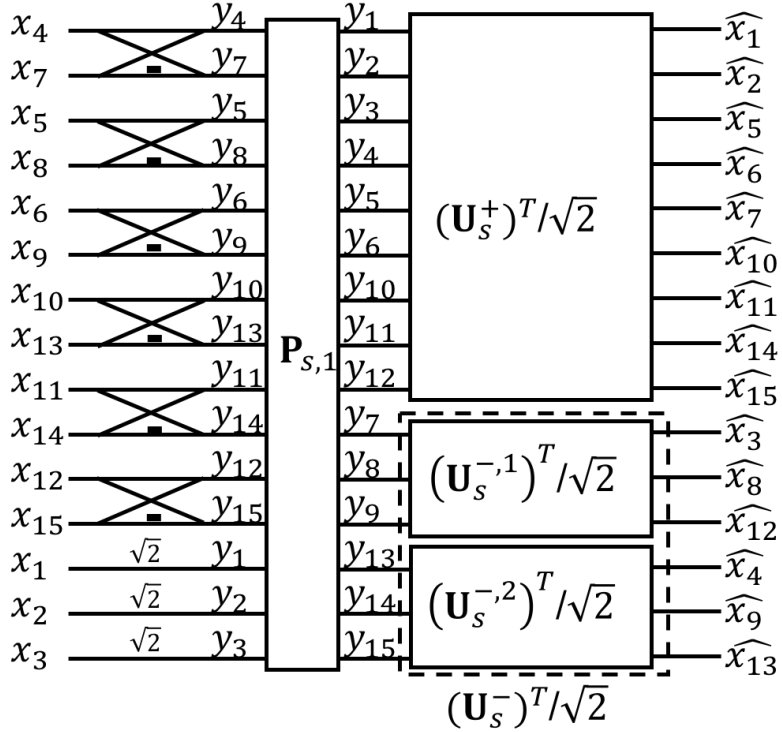
Graph Symmetries

BD Skeleton Graph

Skeleton graph



Transform for Skeleton Graph



Graph Symmetries

Approximation vs Speed

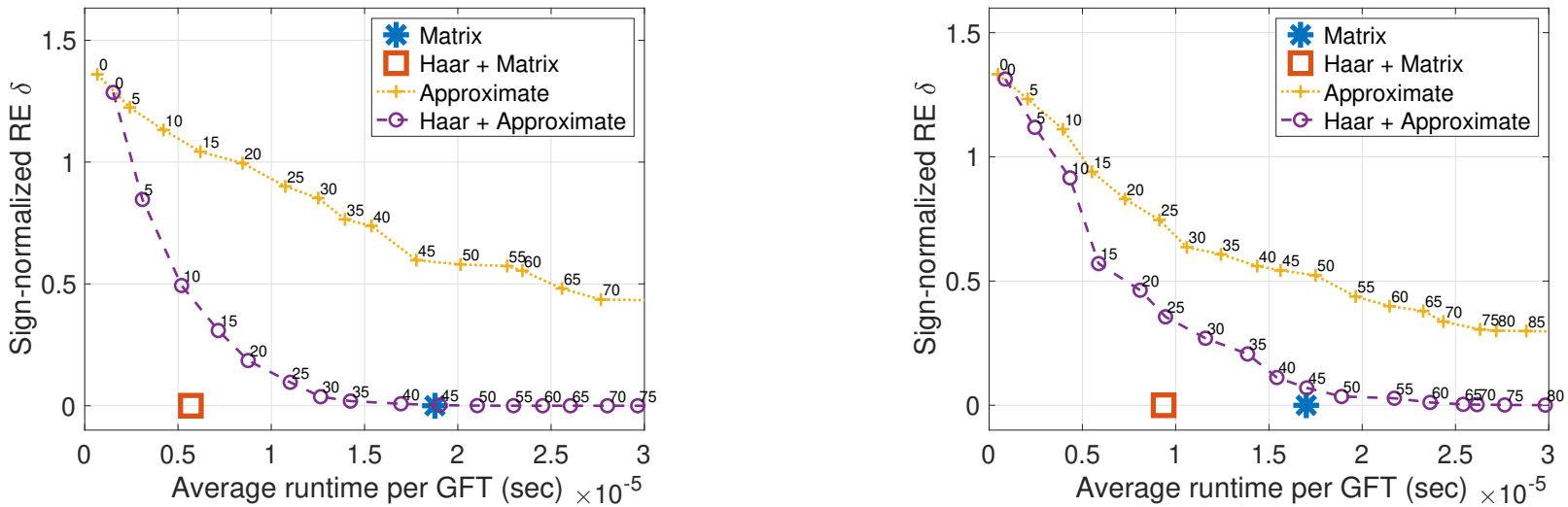


Figure: Runtime versus sign-normalized relative error δ for different GFT implementations on different graphs. 8×8 bi-diagonally symmetric 6 connected grid $\mathcal{G}_b$ (left). 8×8 z-shaped grid $\mathcal{G}_z$ (right). The numbers labeled alongside the markers indicate the corresponding Givens rotation layers.

Clustering

Rank-One Updates & Cauchy Factorizations

Key Idea Partition the graph into subgraphs and build the GFT hierarchically via rank-one Laplacian updates. Each added edge yields a Cauchy factor — enabling an **exact**, divide-and-conquer GFT.

Core Contributions

- **Fast DCT+:** $\mathcal{O}(n \log n)$ GFT for rank-one updates of the path graph — covers DCT-II/III/IV
- **L2G-Net:** Exact GFT factorization $\mathbf{U}^T = \mathbf{D}_K \cdots \mathbf{D}_1 \mathbf{U}_0^T$ via Cauchy factors — scales to $n = 569{,}000$ nodes

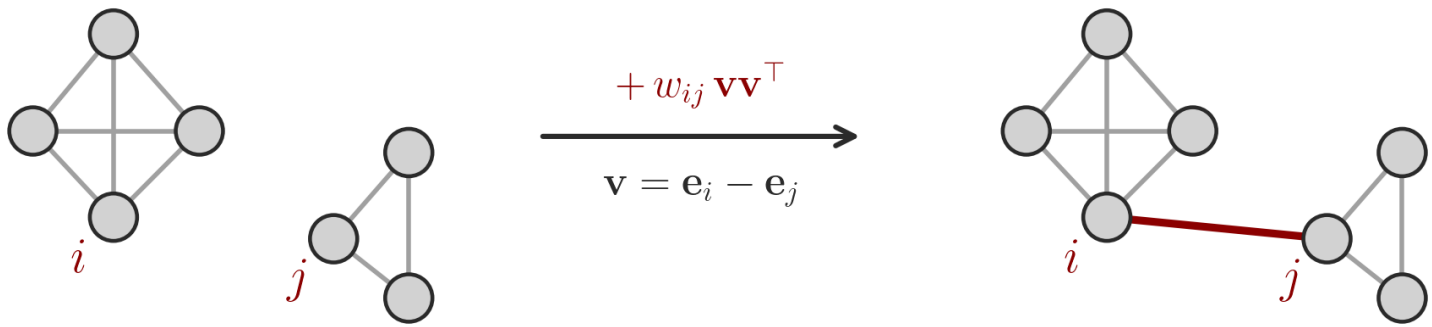
Why This Approach?
Works for any graph — no symmetry required. Exact GFT (no approximation error). Naturally handles graphs built incrementally.

S. Fernández-Menduiña, E. Pavez, A. Ortega, "Fast DCT+: A Family of Fast Transforms Based on Rank-One Updates of the Path Graph", *Proc. ICASSP, IEEE,* 2025
S. Fernández-Menduiña et al., "INT-DTT+: Low-Complexity Data-Dependent Transforms for Video Coding", *Proc. PCS*, 2025
S. Fernández-Menduiña, E. Pavez, A. Ortega, "L2G-Net: Local to Global Spectral Graph Neural Networks via Cauchy Factorizations", *Proc. ICML*, PMLR, 2026

Background: rank-one updates and Cauchy matrices

Adding an edge (i, j) to a graph yields a **rank-one update** of the Laplacian $\mathbf{L}$:

$$\tilde{\mathbf{L}} = \mathbf{L} + w_{ij}\,\mathbf{v}\mathbf{v}^\top, \quad \mathbf{v} = \mathbf{e}_i - \mathbf{e}_j. \tag{1}$$



Rank-one update $\implies$ rotate by an **orthogonal Cauchy-like matrix**:

$$\tilde{\mathbf{U}}^\top = \mathbf{C}(\tilde{\boldsymbol{\lambda}}, \boldsymbol{\lambda})\,\mathbf{U}^\top. \tag{2}$$

Key Observation

Adding a single edge (i, j) with weight w_{ij} is a rank-one update of the Laplacian:

$$\tilde{\mathbf{L}} = \mathbf{L} + w_{ij}\,\mathbf{v}\mathbf{v}^T, \quad \mathbf{v} = e_i - e_j$$

If $\mathbf{L} = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda})\,\mathbf{U}^T$, then:

$$\tilde{\mathbf{U}}^T = \mathbf{C}(\tilde{\boldsymbol{\lambda}}, \boldsymbol{\lambda})\,\mathbf{U}^T$$

where $C_{ij} = \dfrac{a_j z_i}{\tilde{\lambda}_j - \lambda_i}$ — an orthogonal Cauchy-like matrix.

Eigenvalue Interlacing

$$\lambda_1 \leq \tilde{\lambda}_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq \tilde{\lambda}_n$$

Updated eigenvalues found via the *secular equation*:

$$1 + w_{ij} \sum_{i=1}^{n} \frac{z_i^2}{\lambda_i - \lambda} = 0$$

DCT+ Family (Fast DCT+, arXiv:2409.08970)

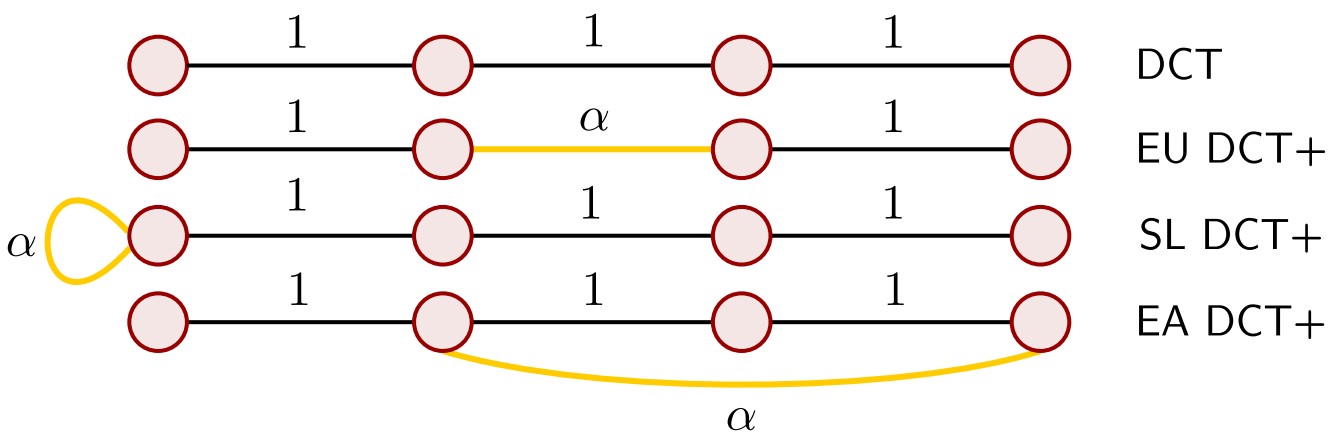
Path graph Laplacian has the DCT-II eigenbasis. A single rank-one update yields a new GFT in the DCT+ family — covering DCT-II, DCT-III, DCT-IV, etc

Fast Algorithm Steps

1. Apply unperturbed GFT $\mathbf{U}^T$ (e.g., fast DCT): $\mathcal{O}(n \log n)$
2. Multiply by Cauchy matrix $\mathbf{C}(\tilde{\boldsymbol{\lambda}}, \boldsymbol{\lambda})$: exploits Cauchy structure for $\mathcal{O}(n \log n)$

Complexity Result

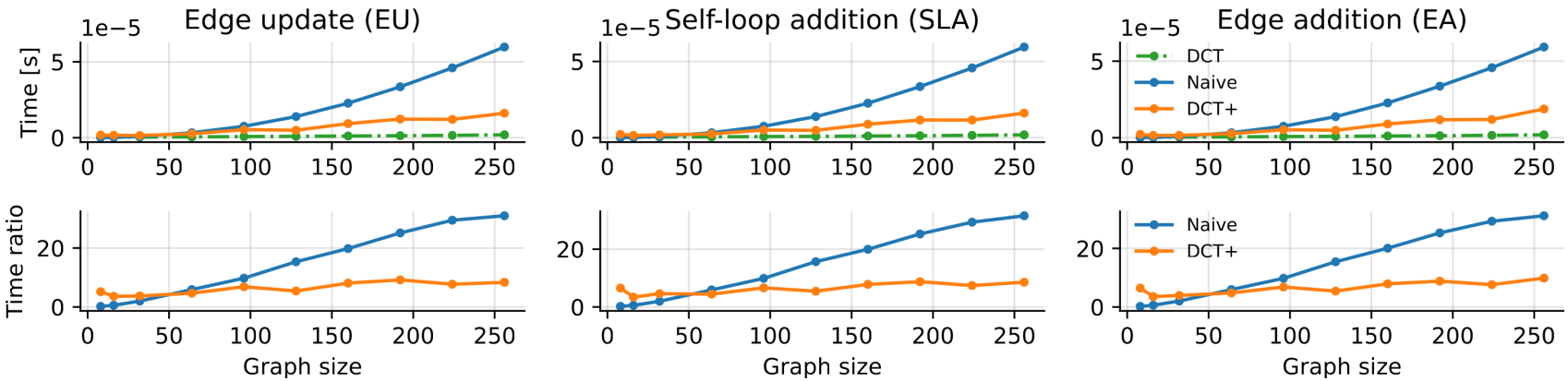
Fast DCT+ achieves $\mathcal{O}(n \log n)$ — same as FFT.



Self-loop: video coding [Han et al., 2011, Zhao et al., 2021].
Edge: Lapped transform [Lu et al., 2019], graph learning [Egilmez et al., 2020].

Clustering

DCT+ Empirical validation



Time ratios with respect to the DCT

Fast DCT+ is more efficient than naive matvec for signals larger than 64.

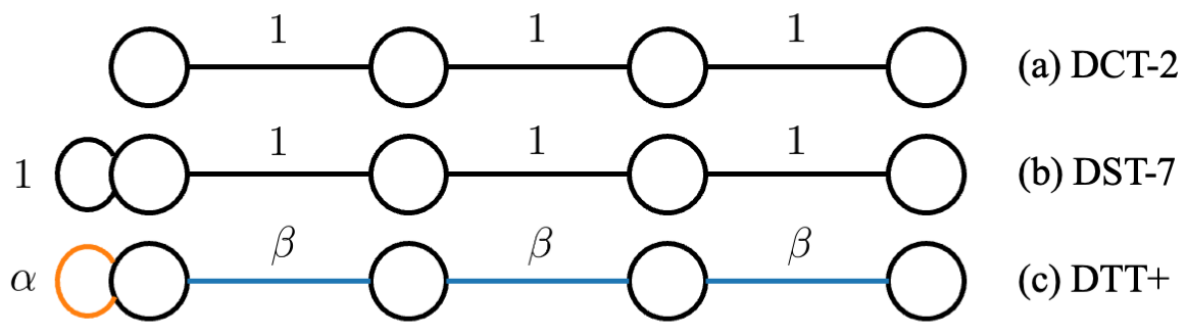
Ratio Fast DCT+/DCT is approximately constant.

Precision was set to 100 dB, which is always satisfied.

Clustering

DTT+

- **DCT-2**: pixel can be predicted from neighbors.
- **DST-7**: boundary pixels can be perfectly predicted.
- **DTT+**: Rank-one updates of the path graph or DTT+.



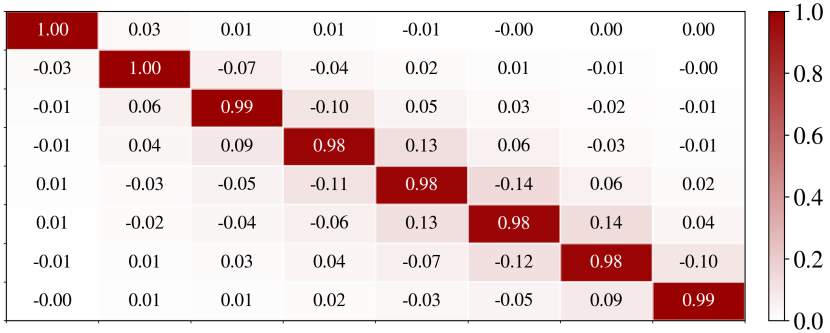
Learnable: Model residuals after prediction and choose (α, β) from examples.

Low complexity: Sparse transformation on top of DTT coefficients.

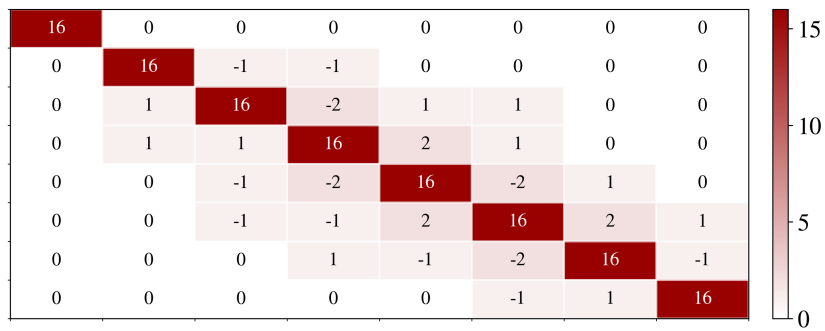
Clustering

Fast products with Cauchy matrices

(a) Learned Cauchy kernel



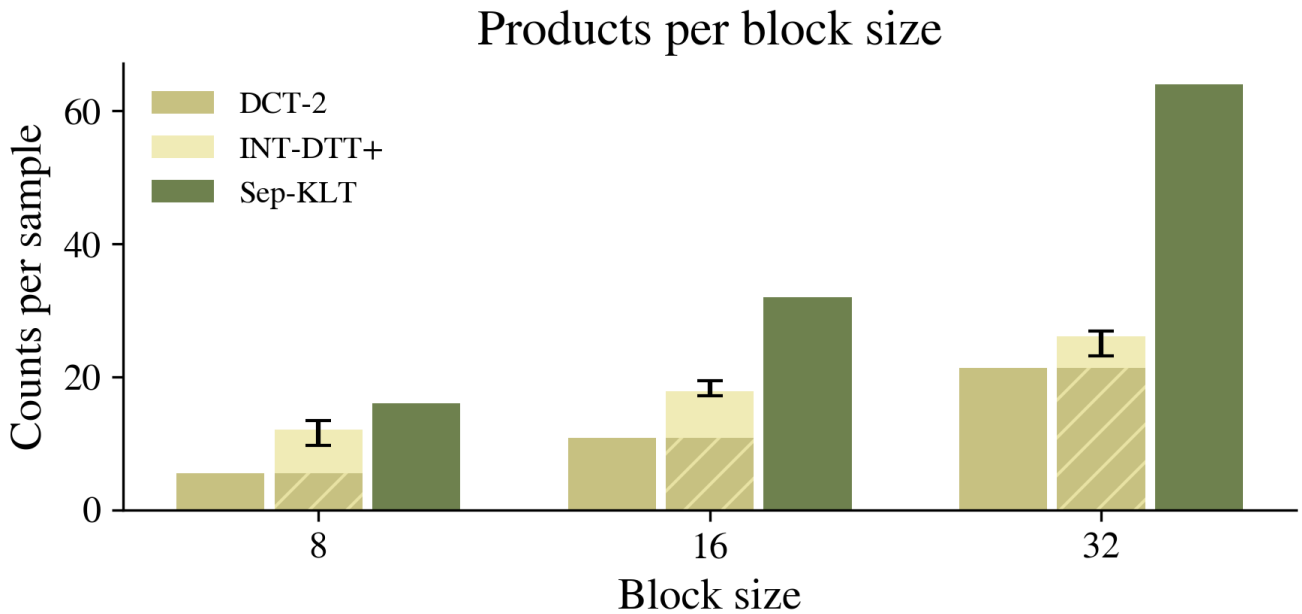
(b) Integer Cauchy kernel



- Main idea: we can ignore the smaller out-of-diagonal coefficients.
- We quantize them to obtain a sparse kernel.
- Reduces the number of products.

Clustering

Coding results: INT-DTT+

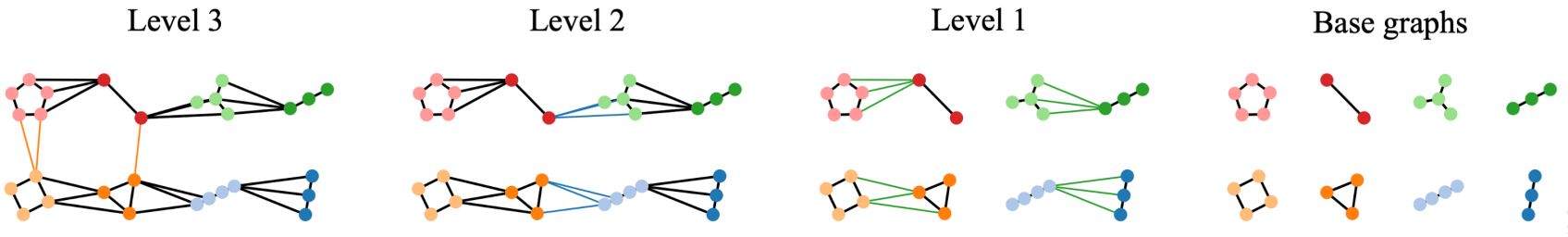


Size	sep-KLT	DTT+	INT-DTT+
8×8	$-\mathbf{3.21}$	-3.12	-3.06
16×16	-3.60	$-\mathbf{3.64}$	-3.46
32×32	-3.72	$-\mathbf{3.96}$	-3.75

Table: BD-rate savings (%) vs VVC-MTS in Y-PSNR. Lower is better.

Clustering

GFT via divide and conquer



1. **Partition** the graph into subgraphs (e.g., balanced cuts).
2. **Compute** GFT of base graphs.
3. Progressively **merge** base GFTs using edge information.

Key: adding one edge adds one Cauchy factor.

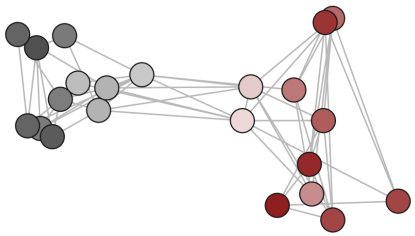
Cauchy Factorization Theorem For graph $\mathcal{G}$ with k bridge edges between m subgraphs $\{\mathcal{G}_i\}$, with $\mathbf{U}_0$ the block-diagonal subgraph eigenvector matrix:

$$\mathbf{U}^T = \mathbf{D}(\boldsymbol{\lambda}, \tilde{\boldsymbol{\lambda}}_{K-1}) \cdots \mathbf{D}(\tilde{\boldsymbol{\lambda}}_1, \tilde{\boldsymbol{\lambda}}_0)\, \mathbf{U}_0^T$$

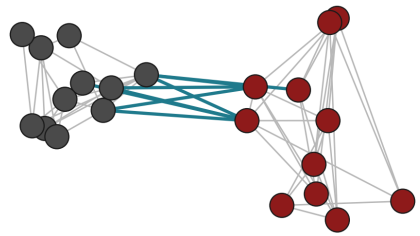
Each Cauchy factor $\mathbf{D}$ is localized to the two subgraphs it connects. The chain computes the **exact** global GFT.

Clustering

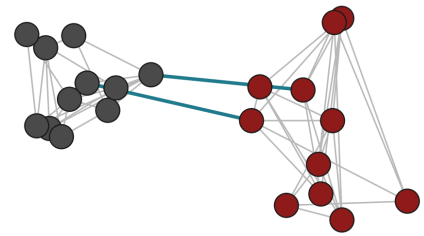
Finding a suitable hierarchy



Fiedler vector



Partition: 8 bridge edges



Sparsified: 2 bridge edges

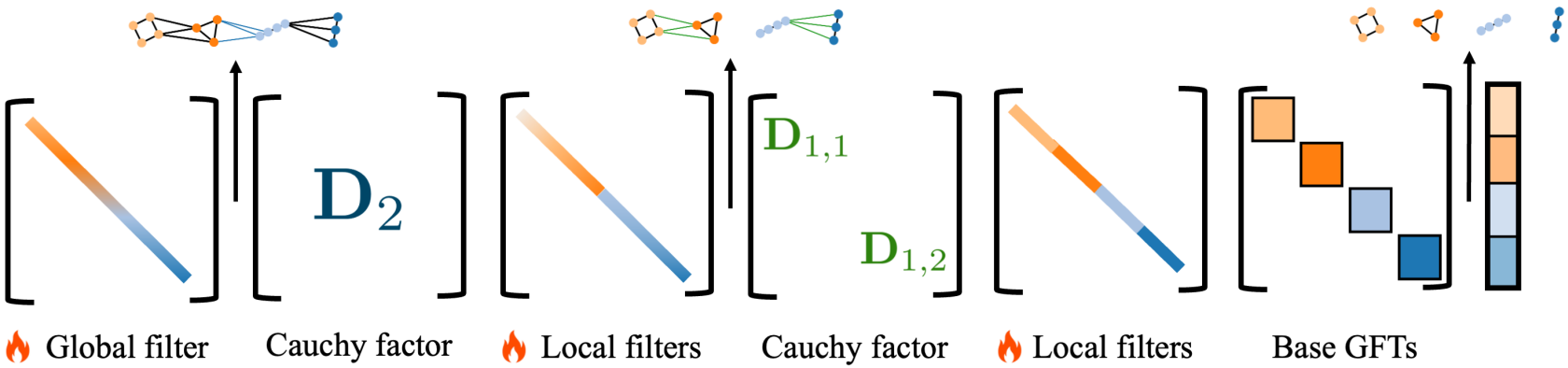
We want small k. **Balanced cuts** via spectral bisection (Fiedler vector).

Accept partition only if theoretical complexity reduces.

Cut sparsification: reduce k via sparsification (Spielman-Srivastava).

Clustering

L2G-Net



Local filters capture patterns within subgraphs.

Global filter models long-range subgraph interactions.

Intuition: architecture **adapts to graph structure** before training begins.

Key Results

- 3–4 $\times$ 10^3 speedup vs. dense eigendecomposition on city-scale graphs
- 4$\times$ memory reduction vs. full GFT
- 569k-node London graph factorized in $\sim$144 min
- 97.50 AUC on Minesweeper (best among all methods)
- $\ll 10^7$ learnable parameters vs. Polynormer 10^6–10^7

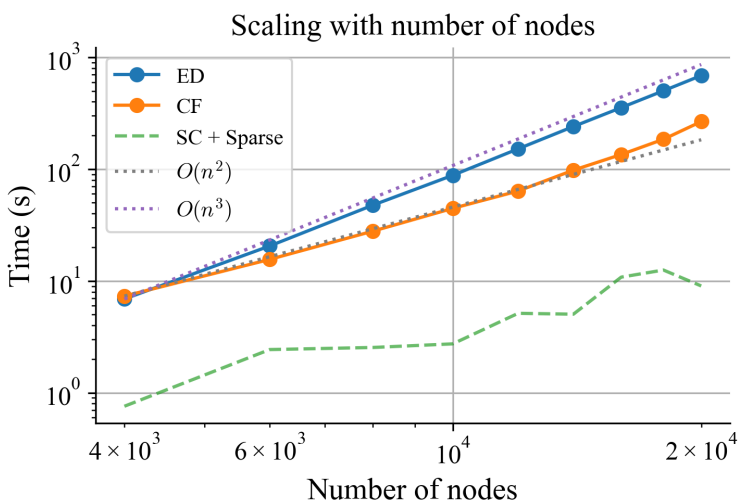
L2G-Net: Design Space Comparison

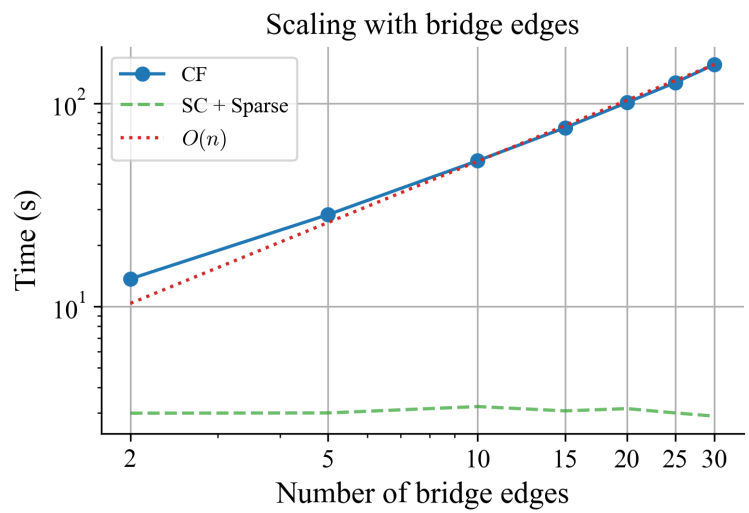
Method	Complexity	Receptive Field
MPNNs	$\mathcal{O}(\lvert\mathcal{E}\rvert)$	Local
GT	$\mathcal{O}(n^2)$	Global
ChebNet	$\mathcal{O}(K\lvert\mathcal{E}\rvert)$	K-hop
Global GFT	$\mathcal{O}(n^3)$	Global
L2G-Net	$\mathcal{O}(kn^2)$	Local$\rightarrow$Global

Runtime Scaling

Cauchy factorization: $\mathcal{O}(n^2)$ at $k = 5$ on Barabási–Albert graphs. Full ED: $\mathcal{O}(n^3)$.

Linear scaling with cut size k (given a fixed $n = 10000$).



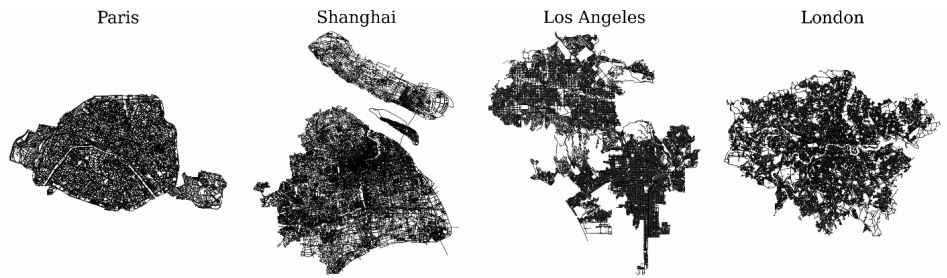


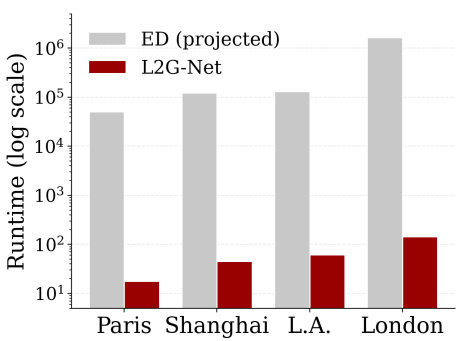
Clustering

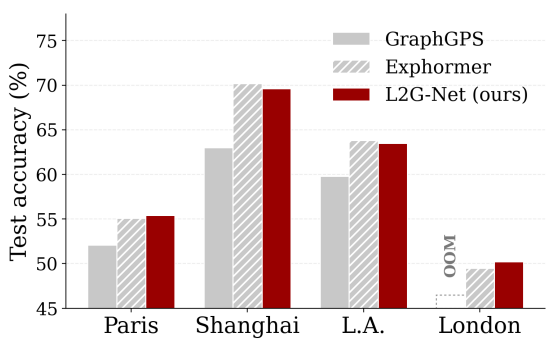
GNNs in CityNetworks

Transductive task: node classification problem in city networks.

Long-rage: Needs local eccentricities of each node.







- Runtime: Scales to graphs where GFT is unfeasible.
- Test accuracy: Matches performance with fewer parameters.

Conclusions & Summary

Four Approaches to Fast GFTs

Approach	Exact?	Class
1. Freq. approx.	$\times$	Yes
2. Vertex approx.	$\approx$	Yes
3. Symmetries	✓	No
4. Clustering	✓	Yes

Key Takeaways

- No single fast GFT algorithm works for all graphs
- Graph structure determines which approach is best
- Exact methods (3, 4) preferred when structure is available
- Approximate methods (1, 2) are graph-agnostic

References

1. Le Magoarou, Gribonval, Tremblay. "Approximate fast graph Fourier transforms via multi-layer sparse approximations." *IEEE Trans. Signal Inf. Process. Netw.*, 2017. arXiv:1612.04542
2. Spielman & Srivastava. "Graph sparsification by effective resistances." *SIAM J. Comput.*, 2011.
3. K.-S. Lu & A. Ortega. "Fast Graph Fourier Transforms Based on Graph Symmetry and Bipartition." *IEEE Trans. Signal Inf. Process. Netw.*, 2019. arXiv:1907.07875
4. S. Fernández-Menduiña, E. Pavez, A. Ortega. "Fast DCT+: A Family of Fast Transforms Based on Rank-One Updates of the Path Graph." 2024. arXiv:2409.08970
5. S. Fernández-Menduiña, E. Pavez, A. Ortega. "L2G-Net: Local to Global Spectral Graph Neural Networks via Cauchy Factorizations." *Proc. 43rd ICML*, PMLR, 2026.
6. A. Cantoni, & Butler, P. "Eigenvalues and eigenvectors of symmetric centrosymmetric matrices". Linear Algebra and its Applications, 13(3), 275-288, 1976.