Algorithm Fast Fourier Transform (FFT)

Scientific computing

Mgr. Markéta Trnečková, Ph.D.

From DFT to FFT – Why Complexity Matters

- **Goal today:** Understand the computational cost of the Discrete Fourier Transform (DFT) and motivate the need for a faster algorithm (FFT).
- **DFT as nested sums** (1D, length N):

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \quad k = 0, \ldots, N-1.$$

  For each output $X[k]$ we sum over all N inputs $\Rightarrow$ N **multiplies** $+$ $N-1$ **adds**.
- **Immediate implication** We must do this for all k (there are N of them) $\Rightarrow$ $\mathcal{O}(N^2)$ operations in general.

Operation Counting for 1D DFT

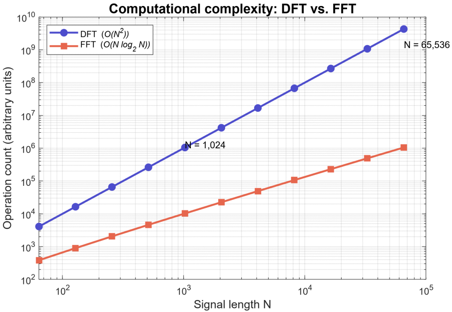
- **Per output bin**
  - N complex multiplications and $(N-1)$ complex additions.
- **For all N outputs**

$$\text{multiplies: } N \cdot N = N^2,$$

$$\text{adds: } N \cdot (N-1) \approx N^2.$$

  Up to constant factors, the 1D DFT is $\mathcal{O}(N^2)$.

- **Concrete numbers (intuition)**
  - $N = 1{,}024$: $\sim 1.0 \times 10^6$ complex multiplies.
  - $N = 65{,}536$: $\sim 4.3 \times 10^9$ complex multiplies.



2D DFT: Naive vs. Separable Cost

- **2D DFT (size $M \times N$)**

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}.$$

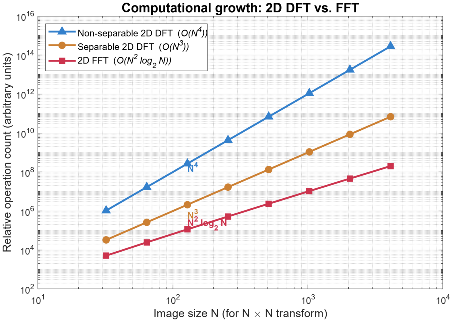
- **Naive (fully nested)**: each (u,v) sums over all MN pixels, and there are MN outputs $\Rightarrow \mathcal{O}(M^2N^2)$.
- **Using separability (still naive 1D DFT)**:
  - Row DFTs: M transforms of length $N \Rightarrow \mathcal{O}(MN^2)$.
  - Column DFTs: N transforms of length $M \Rightarrow \mathcal{O}(NM^2)$.

  $\Rightarrow$ $\mathcal{O}(MN(M+N))$ (still cubic for $M = N$).

Separability means that a multidimensional transform (e.g., 2D DFT) can be computed as successive applications of simpler one-dimensional transforms – for example, first along rows and then along columns.

Square Images: Orders of Growth

- For $N \times N$:
- **Naive 2D**: $\mathcal{O}(N^4)$,
- **Separable (naive 1D)**: $\mathcal{O}(N^3)$
- **2D FFT**: $\mathcal{O}\left(N^2 \log_2 N\right)$
- Even with separability, direct DFTs become quickly impractical for large images

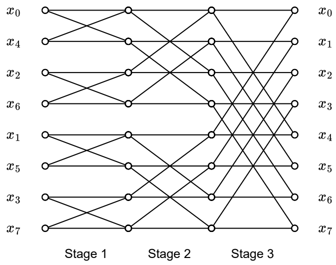


Constant Factors, Memory, and Practical Runtime

- **Not just big-$\mathcal{O}$** Real runtime depends on constant factors (twiddle reuse), memory traffic, cache behavior, and vectorization.
- **Cache-friendly FFT** FFT reorders computation into small "butterflies" that reuse data and reduce cache misses; direct DFT has poor locality.
- **Data sizes** Power-of-two lengths (radix-2) often yield the fastest paths. Mixed-radix and Bluestein algorithms help for prime/awkward sizes.
- **2D/3D transforms** Apply 1D FFTs along rows/cols (and slices), benefiting from separability while keeping good locality.

Reusing Structure

- **Observation** The DFT sums use repeated complex exponentials (twiddle factors). FFTs exploit:
  - **periodicity** and **symmetry** of $e^{-j2\pi kn/N}$,
  - **divide-and-conquer** splits into even/odd (or small radices).
- Large numbers of multiplications become **shared** across outputs, reducing complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.
- We will derive the radix-2 Cooley–Tukey scheme: decimation-in-time (DIT), butterflies, bit-reversal ordering.



DFT Factorization – Decimation in Time (DIT)

- Start from the length-N DFT (assume N even):

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{kn}, \quad W_N = e^{-j\frac{2\pi}{N}}, \quad k = 0, \ldots, N-1.$$

- Split indices into **even** $n = 2m$ and **odd** $n = 2m+1$:

$$X[k] = \sum_{m=0}^{\frac{N}{2}-1} x[2m]\, W_N^{k(2m)} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1]\, W_N^{k(2m+1)}.$$

- Use $W_N^{2k} = W_{N/2}^{k}$:

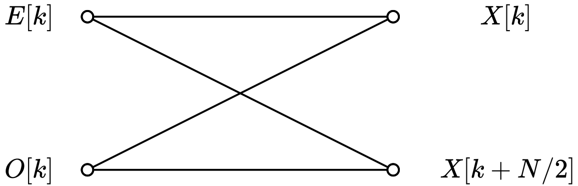
$$X[k] = E[k] + W_N^k\, O[k], \quad E[k] = \sum_m x[2m]\, W_{N/2}^{km}, \quad O[k] = \sum_m x[2m+1]\, W_{N/2}^{km}.$$

Radix-2 DIT: Periodization Over k

- For $k = 0, \ldots, \frac{N}{2} - 1$:

$$X[k] = E[k] + W_N^k\, O[k],$$
$$X[k + \tfrac{N}{2}] = E[k] - W_N^k\, O[k].$$

- One pair $(E[k], O[k])$ produces a **butterfly** that yields two outputs.
- Recursively apply the same idea to compute $E[\cdot]$ and $O[\cdot]$ (each is an FFT of size $N/2$).



Butterfly Mechanics (Complex Adds + Twiddle Multiply)

- A radix-2 **butterfly** maps two complex inputs (a, b) to two outputs:

$$y_0 = a + w\,b, \qquad y_1 = a - w\,b, \quad w = W_N^k.$$

- Work per butterfly: **1 complex multiply** + **2 complex adds**.
- Number of stages: $\log_2 N$. Number of butterflies per stage: $N/2$.
- Total butterflies: $(N/2)\log_2 N \Rightarrow \mathcal{O}(N \log N)$.

Bit-Reversed Ordering and In-Place Execution

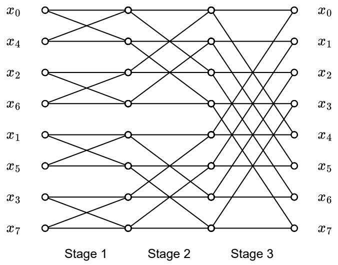
- **In-place FFT**: rearrange samples so that each stage reads/writes locally.
- **Bit reversal**: index n (in binary) is reversed to place inputs in the order needed by the butterfly network.
- Example ($N = 8$): indices 0..7 in binary:

$$000, 001, 010, 011, 100, 101, 110, 111 \;\Rightarrow\; 000, 100, 010, 110, 001, 101, 011, 111.$$

- After bit-reversal, stages operate on **contiguous** pairs, then quartets, etc.

Example (N=8)

- Input $x[0..7]$: $x = [1, 1, 1, 1, 0, 0, 0, 0]$.
- Stage 0 (bit-reverse reorder) $\Rightarrow$ **paired** inputs.
- Stage 1: butterflies on contiguous pairs, twiddle $W_2^0 = W_8^0 = 1$.
- Stage 2: butterflies on groups of 4, twiddles $W_4^0 = W_8^0 = 1$, $W_4^1 = W_8^2 = e^{-j\pi/2}$.
- Stage 3: butterflies on groups of 8, twiddles $W_8^0 = 1$, $W_8^1 = e^{-j\pi/4}$, $W_8^2 = e^{-j\pi/2}$, $W_8^3 = e^{-j3\pi/4}$.



Example (N=8)

- Use $x = [1, 1, 1, 1, 0, 0, 0, 0]$ and compute:

$$\text{After Stage 1: pairwise sums/diffs (no twiddles).}$$
$$\text{After Stage 2: apply } W_8^2 \text{ where needed.}$$
$$\text{After Stage 3: final combine to get } X[0..7].$$

- You should obtain a concentrated low-frequency spectrum (most energy near $k = 0$).

Example – solution

- $x[n] = [1, 1, 1, 1, 0, 0, 0, 0], \quad N = 8$
- Conventions:
  - Forward FFT (no $1/N$ scaling)
  - $W_N = e^{-j2\pi/N}$
  - Bit-reversal ordering applied before Stage 1

Example – solution

Stage 0 – Bit-Reversal Reordering

- For $N = 8$ (3 bits per index), the bit-reversal permutation is:

$$[0, 1, 2, 3, 4, 5, 6, 7] \;\Rightarrow\; [0, 4, 2, 6, 1, 5, 3, 7].$$

- After reordering, the sequence becomes:

$$x^{(0)} = [x_0, x_4, x_2, x_6, x_1, x_5, x_3, x_7] = [1, 0, 1, 0, 1, 0, 1, 0].$$

- This layout ensures that each butterfly at later stages reads contiguous samples.

Example – solution
Stage 1 – Length-2 Butterflies

- Operate on pairs: $(0, 1)$, $(2, 3)$, $(4, 5)$, $(6, 7)$.
- All twiddle factors are $W_8^0 = 1$, so only additions/subtractions occur.
- Each butterfly computes:

$$(a, b) \;\rightarrow\; (a + b,\; a - b).$$

- With all pairs $(1, 0)$, we get:

$$x^{(1)} = [1, 1, 1, 1, 1, 1, 1, 1].$$

Example – solution

Stage 2 — Length-4 Butterflies (N=8, DIT)

- Two groups: [0..3] and [4..7].
- Twiddle factors (global, N=8, m=4):

$$W_8^0 = 1, \qquad W_8^2 = e^{-j\pi/2} = -j.$$

- With $x^{(1)} = [1, 1, 1, 1,\ 1, 1, 1, 1]$, each group yields:

$$\begin{aligned}
x_0' &= x_0 + W_8^0 x_2 = 1 + 1 = 2, \\
x_1' &= x_1 + W_8^2 x_3 = 1 + (-j) \cdot 1 = 1 - j, \\
x_2' &= x_0 - W_8^0 x_2 = 1 - 1 = 0, \\
x_3' &= x_1 - W_8^2 x_3 = 1 - (-j) \cdot 1 = 1 + j.
\end{aligned}$$

- Both groups are identical.

$$x^{(2)} = [\,2,\ 1-j,\ 0,\ 1+j,\ 2,\ 1-j,\ 0,\ 1+j\,].$$

Example – solution

Stage 3 – Length-8 Butterfly (Final Combine)

- Pairs: $(0,4)$, $(1,5)$, $(2,6)$, $(3,7)$.
- Twiddle factors:

$$W_8^0 = 1, \quad W_8^1 = e^{-j\pi/4} = \tfrac{1}{\sqrt{2}}(1-j), \quad W_8^2 = -j, \quad W_8^3 = -\tfrac{1}{\sqrt{2}}(1+j).$$

- Apply $(a, b) \to (a + wb,\ a - wb)$ for each pair:

$$\begin{aligned}
X_0 &= 2 + 1 \cdot 2 = 4, & X_4 &= 2 - 1 \cdot 2 = 0, \\
X_1 &= (1-j) + W_8^1(1-j) \approx 1 - 2.4142j, & X_5 &= (1-j) - W_8^1(1-j) \approx 1 + 0.4142j, \\
X_2 &= 0, & X_6 &= 0, \\
X_3 &= (1+j) + W_8^3(1+j) \approx 1 - 0.4142j, & X_7 &= (1+j) - W_8^3(1+j) \approx 1 + 2.4142j.
\end{aligned}$$

Example – solution
Final Result and Interpretation

$$X = [\,4,\ 1 - 2.4142j,\ 0,\ 1 - 0.4142j,\ 0,\ 1 + 0.4142j,\ 0,\ 1 + 2.4142j\,].$$

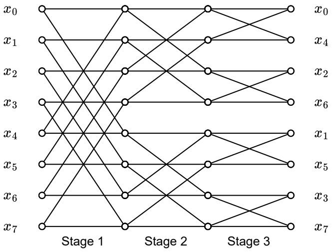
- Most of the energy is concentrated in the DC and low-frequency terms ($k = 0, 1, 7$).
- The symmetry around $k = 0$ and $k = 4$ reflects the real-valued input signal.
- Magnitude spectrum peaks at low k, indicating a slowly varying (almost constant) input.

DIT vs. DIF (and Other Radices)

- **DIT** (decimation in time): split input into even/odd *time* indices; twiddles multiply the odd branch.
- **DIF** (decimation in frequency): split outputs into even/odd *frequency* indices; twiddles multiply one output path.
- Both produce **identical spectra**; differ in dataflow, twiddle placement, and where bit-reversal appears (input vs. output).
- **Other radices**: radix-4, mixed-radix (e.g., $N = 2^a 3^b 5^c$), and Bluestein/Chirp-Z for prime sizes.

Radix-2 DIF FFT – Concept

- **Decimation in Frequency (DIF)**: splits the output spectrum into even and odd frequency indices.
- Each stage performs **butterflies** across larger and larger distances (starting from N).
- **Twiddle factors** multiply the lower (odd-frequency) branch.
- Input remains in **natural order**; bit-reversal occurs on the output.



DIF Butterfly Operation

- For a block of size m, with base index b and offset $r = 0, \ldots, m/2 - 1$:

$$u = x[b + r], \quad v = x[b + r + m/2],$$
$$s = u + v, \qquad d = (u - v)\, W_m^r, \quad W_m = e^{-j\frac{2\pi}{m}},$$
$$x[b + r] \leftarrow s, \quad x[b + r + m/2] \leftarrow d.$$

- **Upper outputs (s)** $\Rightarrow$ even frequency bins.
- **Lower outputs (d)** $\Rightarrow$ odd frequency bins (twiddled).
- Each stage halves the butterfly length: $N \rightarrow N/2 \rightarrow N/4 \rightarrow \cdots \rightarrow 2$.

Example – solution

Stage 1 – Length-8 Butterflies

- Stage size $m = 8$, $r = 0, 1, 2, 3$; pairs $(0, 4)$, $(1, 5)$, $(2, 6)$, $(3, 7)$.
- Twiddles: $W_8^r = e^{-j\frac{2\pi r}{8}}$.
- Input: $x = [1, 1, 1, 1, 0, 0, 0, 0]$.

$$\begin{aligned}
r{=}0 &: u = 1,\ \ v = 0 \Rightarrow s = 1,\ \ d = (1-0)\,W_8^0 = 1,\\
r{=}1 &: u = 1,\ \ v = 0 \Rightarrow s = 1,\ \ d = (1-0)\,W_8^1 = \tfrac{1}{\sqrt{2}}(1-j),\\
r{=}2 &: u = 1,\ \ v = 0 \Rightarrow s = 1,\ \ d = (1-0)\,W_8^2 = -j,\\
r{=}3 &: u = 1,\ \ v = 0 \Rightarrow s = 1,\ \ d = (1-0)\,W_8^3 = -\tfrac{1}{\sqrt{2}}(1+j).
\end{aligned}$$

- After Stage 1:

$$x^{(1)} = \big[\,1,\ 1,\ 1,\ 1,\ 1,\ \tfrac{1}{\sqrt{2}}(1-j),\ -j,\ -\tfrac{1}{\sqrt{2}}(1+j)\,\big].$$

Example – solution

Stage 2 — Two Length-4 Butterflies

- Now process two groups: [0..3] and [4..7], each of length $m = 4$.
- Twiddles: $W_4^r = e^{-j\frac{2\pi r}{4}} = [1, -j]$.
- Upper group: $[1, 1, 1, 1]$

$$r{=}0 : u = 1,\ v = 1 \Rightarrow s = 2,\ d = (1-1)W_4^0 = 0,$$
$$r{=}1 : u = 1,\ v = 1 \Rightarrow s = 2,\ d = (1-1)W_4^1 = 0.$$

- Lower group: $[1,\ \frac{1}{\sqrt{2}}(1-j),\ -j,\ -\frac{1}{\sqrt{2}}(1+j)]$

$$r{=}0 : u = 1,\ v = -j \Rightarrow s = 1-j,\ d = (1+j)W_4^0 = 1+j,$$
$$r{=}1 : u = \tfrac{1}{\sqrt{2}}(1-j),\ v = -\tfrac{1}{\sqrt{2}}(1+j) \Rightarrow s = -\sqrt{2}\,j,$$
$$d = (\tfrac{1}{\sqrt{2}}(1-j) - (-\tfrac{1}{\sqrt{2}}(1+j)))W_4^1 = \sqrt{2}(-j) = -\sqrt{2}\,j.$$

- After Stage 2:

$$x^{(2)} = [\,2,\ 2,\ 0,\ 0,\ 1-j,\ -\sqrt{2}\,j,\ 1+j,\ -\sqrt{2}\,j\,].$$

Example – solution

Stage 3 — Four Length-2 Butterflies

- Stage size $m = 2$, $r = 0$; pairs $(0,1)$, $(2,3)$, $(4,5)$, $(6,7)$.
- $W_2^0 = 1$ (no twiddles in this final stage).

$$\begin{aligned}
(0,1) :&(2,2) \Rightarrow s = 4,\ d = 0,\\
(2,3) :&(0,0) \Rightarrow s = 0,\ d = 0,\\
(4,5) :&(1-j,\ -\sqrt{2}j) \Rightarrow s = 1-(1+\sqrt{2})j,\ d = 1+(\sqrt{2}-1)j,\\
(6,7) :&(1+j,\ -\sqrt{2}j) \Rightarrow s = 1+(1-\sqrt{2})j,\ d = 1+(1+\sqrt{2})j.
\end{aligned}$$

- After Stage 3:

$$x^{(3)} = [\,4,\ 0,\ 0,\ 0,\ 1-2.4142j,\ 1+0.4142j,\ 1-0.4142j,\ 1+2.4142j\,].$$

Bit-Reversal and Final Output

- DIF produces outputs in **bit-reversed** order of frequency indices.
- For $N{=}8$, bit-reversal permutation:

$$\pi = [\,0, 4, 2, 6, 1, 5, 3, 7\,].$$

- Reorder to natural order: $X[k] = x^{(3)}[\pi(k)]$.
- Final result:

$$X = [\,4,\ 1 - 2.4142j,\ 0,\ 1 - 0.4142j,\ 0,\ 1 + 0.4142j,\ 0,\ 1 + 2.4142j\,].$$

FFT and Input Length — Why Powers of Two?

- The **radix-2 FFT** assumes that the input length N can be recursively divided by 2:

$$N = 2^m.$$

- Each stage splits the data into even and odd parts until 2-point butterflies remain.
- This structure gives the elegant complexity:

$$\mathcal{O}(N \log_2 N).$$

- If N is not a power of two, the radix-2 pattern **cannot fully subdivide** – the recursion breaks.
- In practice, this matters for real-world signals or images whose dimensions are not powers of 2.

When N Is Not a Power of Two

- Suppose $N = 10$. We can still define the DFT:

$$X[k] = \sum_{n=0}^{9} x[n] e^{-j2\pi kn/10},$$

  but the radix-2 FFT structure no longer applies directly.
- Options:
  - **Zero padding:** Extend $x[n]$ with zeros to the next power of two ($N = 16$).
  - **Mixed-radix FFT:** Factor N into smaller integers, e.g. $N = 2 \times 5$ and combine smaller FFTs.
  - **Prime-length algorithms:** Use Bluestein's or Rader's algorithm for prime N.
- Complexity remains $\mathcal{O}(N \log N)$, but implementation details differ.

What Does "Radix" Mean in FFT?

- **Radix** $=$ the **base** or **branching factor** of the FFT decomposition.
- In the Cooley–Tukey algorithm, we factorize the transform length:

$$N = R \times M,$$

  where R is the **radix** – the number of subtransforms per stage.
- Each stage processes R-input butterflies and recursively computes smaller DFTs.
- Examples:
  - $R = 2 \Rightarrow$ radix-2 FFT (splits even/odd samples),
  - $R = 3 \Rightarrow$ radix-3 FFT (three-way split),
  - $R = 4 \Rightarrow$ radix-4 FFT (fewer multiplications per stage).
- All achieve $\mathcal{O}(N \log N)$ complexity – but with different constants and structure.

Radix-3

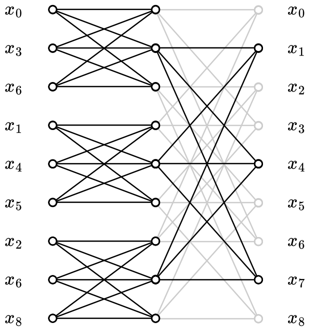
- **Radix-3 FFT:**

  $$X[k] = \sum_{r=0}^{2} x_r[n]\, W_N^{kr}, \quad \text{three branches per butterfly.}$$

  Uses three twiddle factors:

  $$W_3^0 = 1, \quad W_3^1 = e^{-j2\pi/3}, \quad W_3^2 = e^{-j4\pi/3}.$$

- Each stage combines triplets of samples rather than pairs.



Radix-3 Butterfly Equations

- Each 3-point butterfly takes inputs (x_0, x_1, x_2) and produces:

$$\begin{aligned} X_0 &= x_0 + x_1 + x_2, \\ X_1 &= x_0 + W_3^1 x_1 + W_3^2 x_2, \\ X_2 &= x_0 + W_3^2 x_1 + W_3^1 x_2. \end{aligned}$$

- $W_3^1 = e^{-j2\pi/3} = -\frac{1}{2} - j\frac{\sqrt{3}}{2}$, $W_3^2 = e^{-j4\pi/3} = -\frac{1}{2} + j\frac{\sqrt{3}}{2}$.
- Intuition:
  - Each output is a weighted sum of all three inputs,
  - The weights represent complex rotations evenly spaced on the unit circle.

Mixed-Radix FFT – Combining Factors

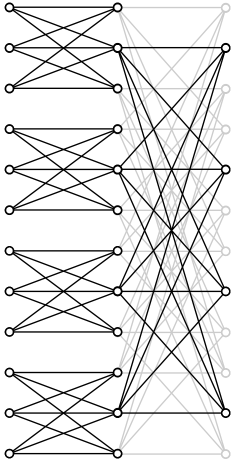
- Real signals rarely have lengths that are perfect powers of two.
- The **mixed-radix FFT** generalizes the idea:

$$N = N_1 \times N_2 \times \dots,$$

  using different radix values for each stage (2, 3, 4, 5, ...).
- Example:

$$N = 12 = 4 \times 3.$$

  First perform three 3-point FFTs (radix-3 stage), then combine with four 4-point FFTs (radix-4 stage).
- Modern libraries (FFTW, NumPy, MATLAB) automatically pick the optimal factorization.

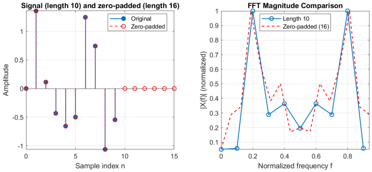


Zero Padding

- **Zero padding** means adding zeros to the end of $x[n]$ before computing the FFT.
- It does **not change the signal's frequency content**, but **increases frequency resolution**.
- The discrete spectrum becomes more densely sampled:

$$X_p[k] = \sum_{n=0}^{N_p-1} x_p[n] e^{-j2\pi kn/N_p}, \quad N_p > N.$$

- Often used for display or interpolation of the magnitude spectrum.
- Be careful: the added resolution is **interpolated**, not additional information.



Mixed-Radix and Prime-Length FFTs

- For non-power-of-two lengths:
  - **Mixed-radix FFT**: Factor N into smaller coprime integers, e.g. $N = 12 = 3 \times 4 \Rightarrow$ compute 3-point and 4-point FFTs and combine.
  - **Radix-3, Radix-5, Radix-7** variants exist for specific factors.
  - **Rader's algorithm**: converts a prime-length FFT into a convolution of length $N - 1$.
  - **Bluestein's algorithm ("chirp-z")**: computes any-length FFT using convolution, useful for arbitrary N.
- Libraries (e.g. FFTW, NumPy) automatically choose the optimal algorithm.
- Complexity: still $\mathcal{O}(N \log N)$, but constants vary depending on factorization.

Practical Recommendations

- **Whenever possible, use power-of-two sizes.**
  - They are fastest, simplest, and best optimized in hardware and software.
- For arbitrary lengths:
  - Zero-pad to the next power of two (if small performance cost is acceptable).
  - Use FFT libraries (e.g. FFTW, NumPy, MATLAB) that automatically pick mixed-radix or Bluestein algorithms.

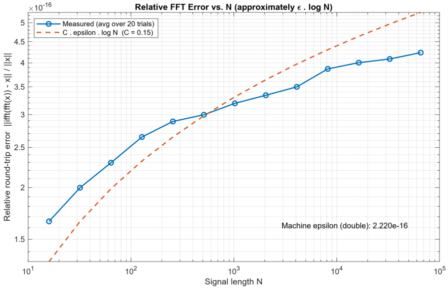
Numerical Aspects of FFT

- **Floating-point arithmetic:** FFT operates on complex numbers with finite precision. Small rounding errors occur in every addition and multiplication.
- **Error accumulation:** Each stage introduces slight numerical noise; overall error grows approximately linearly with $\log_2 N$:

$$\| \hat{X} - X \| \approx \mathcal{O}(\varepsilon \log N),$$

  where ε is the machine precision (typically 10^{-16} in double).

For real-world image sizes, numerical errors are negligible compared to quantization noise and sensor uncertainty.



Numerical Aspects of FFT

- **Numerical stability:** FFT is considered **backward stable** — it gives the exact DFT of slightly perturbed input data.
- **Implementation details:**
  - **In-place computation** minimizes memory traffic.
  - **Cache-friendly radix blocks** (e.g., radix-2, 4, 8) improve performance.
  - Power-of-two sizes yield most efficient memory access and twiddle reuse.

2D FFT – Concept and Definition

- The 2D Discrete Fourier Transform (DFT) for an image $f(x, y)$ of size $M \times N$:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}.$$

- $F(u, v)$ represents the frequency content in both directions:
  - u – horizontal spatial frequency,
  - v – vertical spatial frequency.
- The inverse 2D DFT reconstructs the image:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}.$$

2D FFT – Separable Computation

- The 2D FFT is **separable**:

$$F(u, v) = \mathrm{FFT}_x(\mathrm{FFT}_y(f(x, y))).$$

- Compute FFTs:
  - first along all rows (horizontal 1D FFTs),
  - then along all columns (vertical 1D FFTs).
- Complexity:

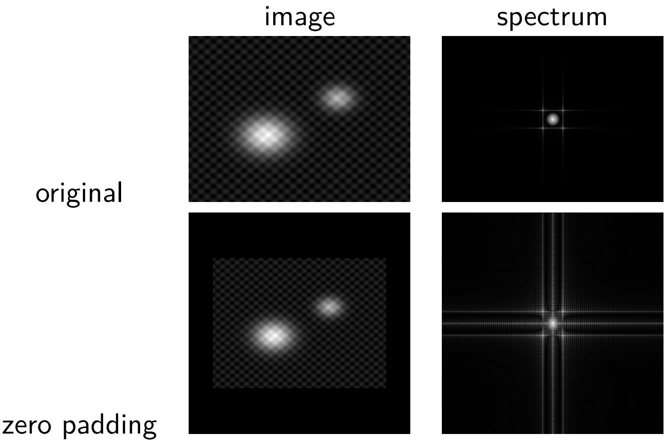
$$\mathcal{O}(MN \log N + MN \log M) \approx \mathcal{O}(MN \log N)$$

  for square images of size $N \times N$.

Zero Padding in 2D FFT

- **Zero padding** means extending the image with zeros to reach a size that is convenient for the FFT algorithm.
- Most FFT implementations (e.g. radix-2) are fastest when each dimension is a power of two:
$$M' = 2^{\lceil \log_2 M \rceil}, \quad N' = 2^{\lceil \log_2 N \rceil}.$$
- Padding to these sizes does not change the image content but allows the FFT to run significantly faster.
- It also increases the sampling density in the frequency domain, making the discrete spectrum appear smoother when visualized.

Zero Padding in 2D FFT



- Zero-padding does not change the frequency content, but it densifies the spectrum samples (providing better visual peak resolution).
- For clarity, the original image is centered within a larger matrix; for FFT-based convolutions you would often embed it in the top-left corner, but centering is more illustrative for demonstration.

Filtering in the Spatial Domain

- Image filtering is most often defined as a **2D convolution**:

$$g(x, y) = (f * h)(x, y) = \sum_{m=-M_h}^{M_h} \sum_{n=-N_h}^{N_h} f(x - m, y - n)\, h(m, n),$$

  where:
  - $f(x, y)$ – input image,
  - $h(m, n)$ – filter kernel (or impulse response),
  - $g(x, y)$ – filtered output.
- This requires $\mathcal{O}(N^2K^2)$ operations for an image of size $N \times N$ and a kernel of size $K \times K$.
- Example filters:
  - Low-pass (Gaussian blur)
  - High-pass (edge enhancement)
  - Directional or sharpening filters

The Convolution Theorem

- The **Convolution Theorem** provides a link between spatial and frequency domains:

$$f * h \;\leftrightarrow \mathrm{FFT}\; F(u,v)\,H(u,v).$$

- Convolution in one domain corresponds to multiplication in the other.
- Implications:
  - Spatial filtering $\rightarrow$ point-wise multiplication in frequency.
  - Frequency shaping (e.g. low-pass mask) $\rightarrow$ convolution in space.
- Therefore, filtering can be computed:
  - either by direct spatial convolution (slow for large kernels),
  - or by FFT multiplication (fast for large images).

Circular Convolution and Wrap-Around Error

- FFT assumes signals and images are **periodic**. Convolution via FFT therefore produces **circular convolution**:

$$g_{\text{circ}}(x, y) = (f * h)_{\text{circular}}.$$

- This causes **wrap-around artifacts**: parts of the filtered image from one edge reappear on the opposite edge.
- To obtain a correct **linear convolution**, we must **pad** both f and h with zeros before transforming:

$$f_p = \text{pad}(f,\ K - 1), \quad h_p = \text{pad}(h,\ N - 1).$$

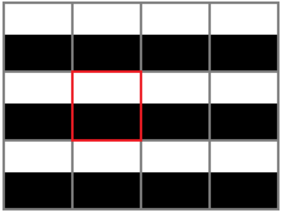
- Safe size (no overlap):

$$M' \geq M + P - 1, \quad N' \geq N + Q - 1,$$

where f has size $M \times N$ and h has size $P \times Q$.

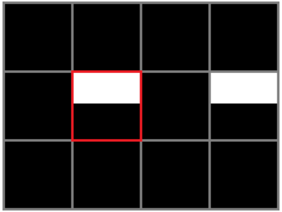
Circular Convolution and Wrap-Around Error

- FFT assumes signals and images are **periodic**. Convolution via FFT therefore produces **circular convolution**:
$$g_{\text{circ}}(x, y) = (f * h)_{\text{circular}}.$$
- This causes **wrap-around artifacts**: parts of the filtered image from one edge reappear on the opposite edge.



Practical Notes for 2D FFT Filtering

- **Padding size:** typically at least $M+P-1$, $N+Q-1$ (or next power of two for speed).



- **Real vs. complex:** real-valued images produce conjugate-symmetric spectra.
- **Boundary artifacts:** use symmetric padding or windowing to minimize edges.

2D Convolution Theorem – FFT-Based Filtering

- Spatial convolution corresponds to multiplication in the frequency domain:

$$f * h \;\leftrightarrow \mathrm{FFT}\; F(u,v)\, H(u,v).$$

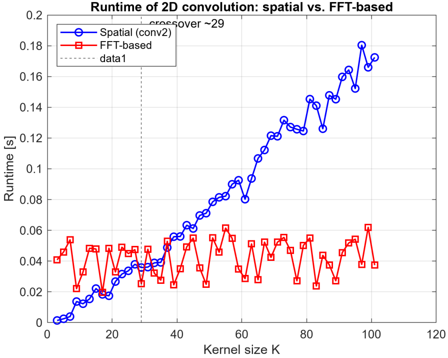
- FFT-based filtering workflow:
  1. Zero-pad both f and h to avoid circular wrap-around ($M' \geq M+P-1$, $N' \geq N+Q-1$).
  2. Compute FFTs: $F = \mathrm{fft2}(f_p)$, $H = \mathrm{fft2}(h_p)$.
  3. Multiply: $G = F \cdot H$.
  4. Inverse FFT: $g = \mathrm{ifft2}(G)$.

2D Convolution Theorem – FFT-Based Filtering

- Computational cost comparison (for $N \times N$ image and $K \times K$ kernel):

$$\text{Direct: } \mathcal{O}(N^2K^2) \quad \text{vs.} \quad \text{FFT-based: } \mathcal{O}(N^2 \log N).$$

- see graph below demonstrating time needed to filter random image with kernels with different sizes



Summary and Key Takeaways

- **Fourier Transform:** decomposes signals or images into frequency components.
- **FFT:** efficient implementation of the DFT with $\mathcal{O}(N \log N)$ complexity.
- **Practical aspects:**
  - Zero-padding to power-of-two sizes for speed and avoiding wrap-around.
  - Numerical accuracy is stable (error grows slowly with $\log N$).
- **Applications in image analysis:**
  - Frequency-domain filtering (low-pass, high-pass, band-pass, see next lesson).
  - Edge enhancement, deblurring, and noise reduction.
- FFT enables fast, large-scale filtering impossible in the spatial domain.