# Fast and Accurate Partial Fourier Transform for Time Series Data

Yong-chan Park
Seoul National University
Seoul, Korea
wjdakf3948@snu.ac.kr

Jun-Gi Jang
Seoul National University
Seoul, Korea
elnino4@snu.ac.kr

U Kang
Seoul National University
Seoul, Korea
ukang@snu.ac.kr

## ABSTRACT

Given a time-series vector, how can we efficiently detect anomalies? A widely used method is to use Fast Fourier transform (FFT) to compute Fourier coefficients, take first few coefficients while discarding the remaining small coefficients, and reconstruct the original time series to find points with large errors. Despite the pervasive use, the method requires to compute *all* of the Fourier coefficients which can be cumbersome if the input length is large or when we need to perform many FFT operations.

In this paper, we propose Partial Fourier Transform (PFT), an efficient and accurate algorithm for computing only a part of Fourier coefficients. PFT approximates a part of twiddle factors (trigonometric constants) using polynomials, thereby reducing the computational complexity due to the mixture of many twiddle factors. We derive the asymptotic time complexity of PFT with respect to input and output sizes, and tolerance. We also show that PFT provides an option to set an arbitrary approximation error bound, which is useful especially when the fast evaluation is of utmost importance. Experimental results show that PFT outperforms the current state-of-the-art algorithms, with an order of magnitude of speedup for sufficiently small output sizes without sacrificing accuracy. In addition, we demonstrate the accuracy and efficacy of PFT on real-world anomaly detection, with interpretations of anomalies in stock price data.

## CCS CONCEPTS

• **Theory of computation** → **Numeric approximation algorithms**; • **Mathematics of computing** → *Time series analysis*; • **Computing methodologies** → *Anomaly detection.*

## KEYWORDS

fast Fourier transform, time series, anomaly detection

**ACM Reference Format:**
Yong-chan Park, Jun-Gi Jang, and U Kang. 2021. Fast and Accurate Partial Fourier Transform for Time Series Data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467293

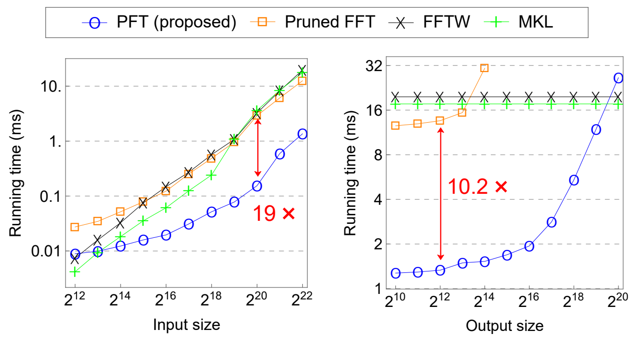
## 1 INTRODUCTION

How can we efficiently compute a specified part of Fourier coefficients of a given time-series vector? Discrete Fourier transform (DFT) is a crucial task in many data mining applications, including anomaly detection [12, 23, 24], latent pattern extraction [22, 26, 34], data center monitoring [18], and image processing [28]. Notably, in many such applications, it is well known that the DFT results in strong "energy compaction" or "sparsity" in the frequency domain. That is, the Fourier coefficients of data are mostly small or equal to zero, having a much smaller support compared to the input size. Moreover, the support can often be specified in practice (e.g., a few low-frequency coefficients around the origin). These observations arouse a great interest in an efficient algorithm which is capable of computing only a specified part of Fourier coefficients. Accordingly, various approaches have been proposed to address the problem, which include Goertzel algorithm [3, 8], Subband DFT [11, 27], and Pruned FFT [1, 16, 19, 29, 31]. However, these methods turn out to be not the successful substitutes for FFT in practice. Indeed, the output size for which they become practical (faster than FFT) is so small compared to the input size that FFT often shows smaller running times even though it computes all the coefficients.

In this paper, we propose Partial Fourier Transform (PFT), an efficient and accurate algorithm for computing a part of Fourier coefficients. Specifically, we consider the following problem: *given a complex-valued vector* $\boldsymbol{a}$ *of size* $N$*, a non-negative integer* $M$*, where* $M \ll N$*, and an integer* $\mu$*, estimate the Fourier coefficients of* $\boldsymbol{a}$ *for the interval* $[\mu - M, \mu + M]$. Our PFT is of remarkably simple structure, composed of several "smaller" FFTs combined with linear pre- and post-processing steps. Consequently, PFT reduces the number of operations from $O(N \log N)$ to $O(N + M \log M)$ which is, to the best of our knowledge, the lowest arithmetic complexity achieved so far. Besides that, most subroutines of PFT are already highly optimized and easy to be parallelized (e.g., matrix multiplication and FFT), thus the arithmetic gains are readily turned into actual run-time improvements. Note that PFT is an approximate method. However, the Fourier coefficients can be evaluated to arbitrary numerical precision with PFT by changing a hyperparameter, trading off running time and error. Experimental results show that PFT outperforms the state-of-the-art FFT libraries, FFTW [7] and Intel Math Kernel Library as well as Pruned FFTW, with an order of magnitude of speedup *without* sacrificing accuracy. We also show the accuracy and efficacy of PFT on real-world anomaly detection tasks, and present interesting discoveries on how an anomalous point in stock price data is related to a real-life incident.

Our main contributions are summarized as follows:

- **Algorithm.** We propose PFT, a novel algorithm for computing a part of Fourier coefficients, achieving the lowest time complexity.
- **Analysis.** We provide theoretical analysis on the arithmetic complexity of PFT and its approximation bound.



(a) Running time vs. input size (b) Running time vs. output size

**Figure 1: (a) Running time vs. input size for target range $R_{0,2^9}$ with $\{S_n\}_{n=12}^{22}$ datasets, and (b) running time vs. output size for fixed input size $2^{22}$. We set the precision of all the methods the same, by making the relative error strictly less than $10^{-6}$. Note that PFT significantly outperforms competitors when the output size is smaller than the input size, which is exactly our target task.**

- **Speed and Accuracy.** PFT shows state-of-the-art speed on both synthetic and real-world datasets without sacrificing accuracy (see Figure 1).
- **Application.** We conduct anomaly detection on real-world datasets, and present discovery results verifying that our algorithm accurately and quickly finds anomalies.

Table 1 summarizes the symbols used. The code and datasets are available at **https://github.com/snudatalab/PFT**.

## 2 RELATED WORKS

We describe the related works on this paper.

### 2.1 Existing Methods

We describe various existing approaches for computing partial Fourier coefficients, and compare them with our proposed method.

**Fast Fourier transform.** One may consider just using Fast Fourier transform (FFT) and discarding the unnecessary coefficients, where FFT efficiently computes the full DFT, reducing the arithmetic cost from naïve $O(N^2)$ to $O(N \log N)$. Such an approach has two major advantages: (1) it is straightforward to implement, and (2) the method often outperforms the competitors because it directly employs FFT which has been highly optimized over decades. Therefore, we provide extensive comparisons of PFT and FFT both theoretically and experimentally. In particular, we show that PFT outperforms the FFT by orders of magnitude when the output size is small enough ($< 10\%$) compared to the input size (Section 4.2).

**Goertzel algorithm.** Goertzel algorithm [3, 8] is one of the first methods devised for computing only a part of Fourier coefficients. The technique is essentially the same as computing the individual coefficients of DFT, thus requiring $O(MN)$ operations for $M$ coefficients of an input with size $N$. Specifically, theoretical analysis represents "the $M$ at which the Goertzel algorithm is advantageous over FFT" as $M < 2 \log N$ [32]. For example, with $N = 2^{22}$, the Goertzel algorithm becomes faster than FFT only when $M < 44$, while PFT outperforms FFT for $M < 2^{19} = 524288$ (Figure 1(b)).

**Table 1: Table of symbols.**

| Symbol | Definition |
|---|---|
| $\omega_n$ | $n$-th primitive root of unity |
| $\hat{\boldsymbol{a}}$ | Fourier coefficient of $\boldsymbol{a}$ |
| $\mathcal{E}(\hat{\boldsymbol{a}})$ | estimated Fourier coefficient of $\boldsymbol{a}$ |
| $N$ | input size descriptor |
| $M$ | output size descriptor |
| $\mu$ | center of target range |
| $R_{\mu,M}$ | target range $[\mu - M, \mu + M] \cap \mathbb{Z}$ |
| $p, q$ | divisors of $N$ |
| $r$ | number of approximating terms |
| $\epsilon$ | tolerance |
| $\|\cdot\|_R$ | uniform norm restricted to $R \subseteq \mathbb{R}$ |
| $P_\alpha$ | set of polynomials on $\mathbb{R}$ of degree at most $\alpha$ |
| $\mathcal{P}_{\alpha,\xi,u}$ | best polynomial approximation to $e^{uix}$ out of $P_\alpha$ under restriction $\lvert x\rvert \leq \lvert\xi\rvert$ |
| $\xi(\epsilon, r)$ | $\sup\{\xi \geq 0 : \|\mathcal{P}_{r-1,\xi,\pi}(x) - e^{\pi ix}\|_{\lvert x\rvert \leq \xi} \leq \epsilon\}$ |
| $w_{\epsilon,r,j}$ | $j$-th coefficient of $\mathcal{P}_{r-1,\xi(\epsilon,r),\pi}$ |

A few variants which improve the Goertzel algorithm have been proposed [2]. Nevertheless, the performance gain is only by a small constant factor, thus they are still limited to rare scenarios where a very few number of coefficients are required.

**Subband DFT.** Subband DFT [11, 27] consists of two stages of algorithm: Hadamard transform that decomposes the input sequence into a set of smaller subsequences, and correction stage for recombination. The algorithm approximates a part of coefficients by eliminating subsequences with small energy contribution, and manages to reduce the number of operations to $O(N + M \log N)$. Apart from the arithmetic gain, however, there is a substantial issue of low accuracy with the Subband DFT. Indeed, experimental results [11] show that the relative approximation error of the method is around $10^{-1}$ (only one significant figure) regardless of output size. Moreover, with PFT, the Fourier coefficients can be evaluated to arbitrary numerical precision, which is not the case for Subband DFT. Such limitations often preclude one from considering the Subband DFT in applications that require a certain degree of accuracy.

**Pruned FFT.** FFT pruning [1, 16, 19, 29, 31] is another technique for the efficient computation of partial Fourier coefficients. The method is a modification of the standard split-radix FFT, where the edges (operations) in a flow graph are pruned away if they do not affect the specified range of frequency domain. Besides being almost optimized (it uses FFT as a subroutine), the FFT pruning algorithm is exact and reduces the arithmetic cost to $O(N \log M)$. Thus, along with the full FFT, the pruned FFT is reasonably the most appropriate competitor of our proposed method. Through experiments (Section 4.2), we show that PFT consistently outperforms the pruned FFT, significantly extending the range of output sizes for which partial Fourier transform becomes practical.

Finally, we mention that there have been other approaches but with different settings. For example, [9, 10] and [13] propose Sparse Fourier transform, which estimates the top-$k$ (the $k$ largest in magnitude) Fourier coefficients of a given vector. The algorithm is useful when there is prior knowledge of the number of non-zero coefficients in frequency domain. Note that our setting does not require any prior knowledge of the given data.

## 2.2 Applications of FFT

We outline various applications of Fast Fourier transform, to which partial Fourier transform can potentially be applied. FFT has been widely used for anomaly detection [12, 23, 24]. [12] and [24] detect anomalous points of a given data by extracting a compact representation using Spectral Residual (SR) based on FFT. [23] uses FFT to detect local spatial outliers which have similar patterns within a region but different patterns from the outside. FFT also serves as the basis for discovering latent patterns [22, 26, 34]. [34] employs FFT to model multi-frequency trading patterns from past market data for stock price prediction. [26] presents an FFT-based approach for automatic sleep stage scoring and apnea-hypopnea detection by extracting meaningful features from EGG, ECG, EOG and EMG data. [22] attempts to detect fake news by designing a CNN-based network that captures the complex patterns of fake images in the frequency domain. Several works [15, 20, 21, 33] exploit FFT for efficient operations. [20] leverages FFT to efficiently compute a polynomial kernel used with support vector machines (SVMs). [15] proposes an efficient Tucker decomposition method using FFT, and [33] applies FFT to a circulant linear system for seasonal-trend decomposition. In addition, FFT has been used for fast training of convolutional neural networks [17, 25] and an efficient recommendation model on a heterogeneous graph [14].

## 3 PROPOSED METHOD

We propose PFT, an efficient and accurate algorithm for computing a specified part of Fourier coefficients. The main challenges and our approaches are as follows:

(1) **How can we extract essential information for a specified output?** Considering that only a specified part of coefficients are needed, we should find an algorithm requiring fewer operations than the direct use of conventional FFT. This is achievable by carefully modifying the Cooley-Tukey algorithm, finding *smooth* twiddle factors (trigonometric factors), and approximating them using polynomials (Section 3.1.1).
(2) **How can we reduce approximation cost?** The approach given above involves an approximation process, which would be computationally demanding. We propose using a *base exponential function*, by which all data-independent factors can be precomputed, enabling one to bypass the approximation problem during the run-time (Sections 3.1.2 and 3.2).
(3) **How can we further reduce numerical computation?** We carefully reorder the operations and factorize the terms in order to alleviate the complexity of PFT. Such techniques separate all data-independent factors from data-dependent factors, allowing further precomputation. The arithmetic cost of the resulting algorithm is $O((N+M\log M)\log(1/\epsilon))$, where $N$ and $M$ are input and output size descriptors, respectively, and $\epsilon$ is a tolerance (Sections 3.3 and 3.4).

### 3.1 Approximation of Twiddle Factors

The key of our algorithm is to approximate a part of smooth (less oscillatory) twiddle factors by using polynomial functions, reducing the computational complexity of DFT due to the mixture of many twiddle factors. Using polynomial approximation also allows one to carefully control the degree of polynomial (or the number of approximating terms), enabling fine-tuning the output range and the approximation bound of the estimation. Our first goal is to find a collection of twiddle factors with small oscillations. This can be achieved by slightly adjusting the summand of DFT and splitting the summation as in the Cooley-Tukey algorithm (Section 3.1.1). Next, using a proper base exponential function, we give an explicit form of approximation to the twiddle factors (Section 3.1.2).

*3.1.1 Smooth Twiddle Factors.* Recall that the DFT of a complex-valued vector $\boldsymbol{a}$ of size $N$ is defined as follows:

$$\hat{a}_m = \sum_{n\in[N]} a_n \omega_N^{mn}, \tag{1}$$

where $\omega_N = e^{-2\pi i/N}$ is the $N$-th primitive root of unity, and $[N]$ denotes $\{0, 1, \cdots, N-1\}$ (in this paper, we follow the convention of viewing a vector $\boldsymbol{v} = (v_0, v_1, \cdots, v_{N-1})$ of size $N$ as a finite sequence defined on $[N]$). Assume that $N = pq$ for two integers $p, q > 1$. The Cooley-Tukey algorithm [5] re-expresses (1) as

$$\hat{a}_m = \sum_{k\in[p]}\sum_{l\in[q]} a_{qk+l}\omega_N^{m(qk+l)} = \sum_{k\in[p]}\sum_{l\in[q]} a_{qk+l}\omega_N^{ml}\omega_p^{mk}, \tag{2}$$

yielding two collections of twiddle factors, namely $\{\omega_N^{ml}\}_{l\in[q]}$ and $\{\omega_p^{mk}\}_{k\in[p]}$. Consider the problem of computing $\hat{a}_m$ for $-M \le m \le M$, where $M \le N/2$ is a non-negative integer. In this case, note that the exponent of $\omega_N^{ml} = e^{-2\pi iml/N}$ ranges from $-2\pi iM(q-1)/N$ to $+2\pi iM(q-1)/N$ and the exponent of $\omega_p^{mk} = e^{-2\pi imk/p}$ ranges from $-2\pi iM(p-1)/p$ to $+2\pi iM(p-1)/p$. Here $\frac{(q-1)/N}{(p-1)/p} \sim \frac{1}{p}$, meaning that the first collection contains smoother twiddle factors compared to the second one. Typically, a smoother function results in a better approximation via polynomials. In this sense, it is reasonable to approximate the first collection of twiddle factors in (2) with polynomial functions, thereby reducing the complexity of the computation due to the mixture of two collections of twiddle factors. Indeed, one can further reduce the complexity of approximation: we slightly adjust the summand in (2) as follows.

$$\hat{a}_m = \omega_{2p}^m \sum_{k\in[p]}\sum_{l\in[q]} a_{qk+l}\omega_N^{m(l-q/2)}\omega_p^{mk}. \tag{3}$$

In (3), we observe that the range of exponents of the first collection $\{\omega_N^{m(l-q/2)}\}_{l\in[q]}$ of twiddle factors is $[-\pi iM/p, +\pi iM/p]$, a contraction by a factor of around 2 when compared with the previous $[-2\pi iM(q-1)/N, +2\pi iM(q-1)/N]$, hence even smoother twiddle factors. There is an extra twiddle factor $\omega_{2p}^m$ in (3). Note that, however, it depends on neither $k$ nor $l$, so the amount of the additional computation is relatively small.

*3.1.2 Base Exponential Function.* The first collection of twiddle factors in (3) consists of $q$ distinct exponential functions. One can apply approximation process for each function in the collection; however, this would be time-consuming. A more plausible approach is to 1) choose a base exponential function $e^{uix}$ for a fixed $u \in \mathbb{R}$, 2) approximate $e^{uix}$ using a polynomial, and 3) exploit a property of exponential functions: the laws of exponents. Specifically, suppose that we obtained a polynomial $\mathcal{P}(x)$ that approximates $e^{uix}$ on $|x| \le |\xi|$, where $u, \xi \in \mathbb{R} \setminus \{0\}$. Consider another exponential function $e^{vix}$, where $v \ne 0$. Since $e^{vix} = e^{ui(vx/u)}$, the re-scaled polynomial $\mathcal{P}(vx/u)$ approximates $e^{vix}$ on $|x| \leq |u\xi/v|$. This observation indicates that once we find an approximation $\mathcal{P}$ to $e^{uix}$ on $|x| \leq |\xi|$ for properly selected $u$ and $\xi$, all elements belonging to $\{\omega_N^{m(l-q/2)} = e^{-2\pi im(l-q/2)/N}\}_{l\in[q]}$ can be approximated by re-scaling $\mathcal{P}$. Fixing a base exponential function also enables precomputing a polynomial that approximates it, so that one can bypass the approximation problem during the run-time. We further elaborate this idea in a rigorous manner after giving a few definitions (see Definitions 3.1 and 3.2).

Let $\|\cdot\|_R$ be the uniform norm (or supremum norm) restricted to a set $R \subseteq \mathbb{R}$, that is, $\|f\|_R = \sup\{|f(x)| : x \in R\}$ and $P_\alpha$ be the set of polynomials on $\mathbb{R}$ of degree at most $\alpha$.

**Definition 3.1.** Given a non-negative integer $\alpha$ and non-zero real numbers $\xi, u$, we define a polynomial $\mathcal{P}_{\alpha,\xi,u}$ as the best approximation to $e^{uix}$ out of the space $P_\alpha$ under the restriction $|x| \leq |\xi|$:

$$\mathcal{P}_{\alpha,\xi,u} := \underset{P\in P_\alpha}{\arg\min} \|P(x) - e^{uix}\|_{|x|\leq|\xi|},$$

and $\mathcal{P}_{\alpha,\xi,u} = 1$ when $\xi = 0$ or $u = 0$. □

[30] proved the unique existence of $\mathcal{P}_{\alpha,\xi,u}$, and a few techniques called *minimax approximation algorithms* for computing the polynomial are reviewed in [6].

**Definition 3.2.** Given a tolerance $\epsilon > 0$ and a positive integer $r \geq 1$, we define $\xi(\epsilon, r)$ to be the scope about the origin such that the exponential function $e^{\pi ix}$ can be approximated by a polynomial of degree less than $r$ with approximation bound $\epsilon$:

$$\xi(\epsilon, r) := \sup\{\xi \geq 0 : \|\mathcal{P}_{r-1,\xi,\pi}(x) - e^{\pi ix}\|_{|x|\leq\xi} \leq \epsilon\}.$$

We express the corresponding polynomial as $\mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(x) = \sum_{j\in[r]} w_{\epsilon,r,j} \cdot x^j$. □

In Definition 3.2, we choose $e^{\pi ix}$ as a base exponential function. The rationale behind is as follows. First, using a minimax approximation algorithm, we precompute $\xi(\epsilon, r)$ and $\{w_{\epsilon,r,j}\}_{j\in[r]}$ for several tolerance $\epsilon$'s (e.g. $10^{-1}, 10^{-2}, \cdots$) and positive integer $r$'s (typically $r \leq 25$). When $N, M, p$ and $\epsilon$ are given, we find the minimum $r$ satisfying $\xi(\epsilon, r) \geq M/p$. Then, by the preceding argument, it follows that the re-scaled polynomial function $\mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(-2x(l - q/2)/N)$ approximates $e^{-2\pi ix(l-q/2)/N}$ on $|x| \leq |\frac{N}{2(l-q/2)} \cdot \frac{M}{p}|$ for each $l \in [q]$ (note that if $l - q/2 = 0$, we have $|\frac{N}{2(l-q/2)} \cdot \frac{M}{p}| = \infty$). Here $|\frac{N}{2(l-q/2)} \cdot \frac{M}{p}| = |\frac{q}{2l-q} \cdot M| \geq M$ for all $l \in [q]$. Therefore, we obtain a polynomial approximation on $|m| \leq M$ for each twiddle factor in $\{\omega_N^{m(l-q/2)}\}_{l\in[q]}$, namely $\{\mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(-2m(l - q/2)/N)\}_{l\in[q]}$. Then, from (3),

$$\omega_{2p}^m \sum_{k\in[p]} \sum_{l\in[q]} a_{qk+l}\, \mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(-2m(l - q/2)/N)\omega_p^{mk} \tag{4}$$

gives an estimation of the coefficient $\hat{a}_m$ for $-M \leq m \leq M$.

## 3.2 Arbitrarily Centered Target Ranges

In the previous section, we have focused on the problem of calculating $\hat{a}_m$ for $m$ belonging to $[-M, M]$. We now consider a more general case: let us use the term **target range** to indicate the range where the Fourier coefficients should be calculated, and $R_{\mu,M}$ to denote $[\mu - M, \mu + M] \cap \mathbb{Z}$, where $\mu \in \mathbb{Z}$. Note that the previously given method works only when our target range is centered at $\mu = 0$. A slight modification of the algorithm allows the target range to be arbitrarily centered. One possible approach is as follows: given a complex-valued vector $\boldsymbol{x}$ of size $N$, we define $\boldsymbol{y}$ as $y_n = x_n \cdot \omega_N^{\mu n}$. Then, the Fourier coefficients of $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfy the following relationship (called Shift Theorem):

$$\hat{y}_m = \sum_{n\in[N]} x_n \omega_N^{\mu n} \omega_N^{mn} = \sum_{n\in[N]} x_n \omega_N^{(m+\mu)n} = \hat{x}_{m+\mu}.$$

Thus, the problem of calculating $\hat{x}_m$ for $m \in R_{\mu,M}$ is equivalent to calculating $\hat{y}_m$ for $m \in R_{0,M}$, to which our previous method can be applied. This technique, however, requires extra $N$ multiplications due to the computation of $\boldsymbol{y}$. A better approach, where one can bypass the extra process during the run-time, is to exploit the following lemma.

Lemma 1. *Given a non-negative integer $\alpha$, non-zero real numbers $\xi, u$, and a real number $\mu$, the following equality holds:*

$$e^{ui\mu} \cdot \mathcal{P}_{\alpha,\xi,u}(x - \mu) = \underset{P\in P_\alpha}{\arg\min} \|P(x) - e^{uix}\|_{|x-\mu|\leq|\xi|}. \quad □$$

Proof. See Supplement A.1. □

This observation implies that, in order to obtain a polynomial approximating $e^{uix}$ on $|x - \mu| \leq |\xi|$, we first find a polynomial $\mathcal{P}$ approximating $e^{uix}$ on $|x| \leq |\xi|$, then translate $\mathcal{P}$ by $-\mu$ and multiply it with the scalar $e^{ui\mu}$. Applying this process to the previously obtained approximation polynomials (see Section 3.1.2) yields $\{\omega_N^{\mu(l-q/2)} \cdot \mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(-2(m - \mu)(l - q/2)/N)\}_{l\in[q]}$. We substitute them for $\{\omega_N^{m(l-q/2)}\}_{l\in[q]}$ in (3), which gives the following estimation of $\hat{a}_m$ for $m \in R_{\mu,M}$, where $k \in [p], l \in [q]$, and $j \in [r]$:

$$\begin{aligned}
&\omega_{2p}^m \sum_{k,l} a_{qk+l}\, \omega_N^{\mu(l-q/2)} \mathcal{P}_{r-1,\xi(\epsilon,r),\pi}(-2(m - \mu)(l - q/2)/N)\, \omega_p^{mk} \\
&= \omega_{2p}^m \sum_{k,l} a_{qk+l}\, \omega_N^{\mu(l-q/2)} \sum_j w_{\epsilon,r,j}(-2(m - \mu)(l - q/2)/N)^j \omega_p^{mk} \\
&= \omega_{2p}^m \sum_j \sum_{k,l} a_{qk+l}\, \omega_N^{\mu(l-q/2)}\, w_{\epsilon,r,j}((m - \mu)/p)^j (1 - 2l/q)^j \omega_p^{mk}.
\end{aligned} \tag{5}$$

## 3.3 Efficient Summations

We have found that three main summation steps (each being over $j, k$, and $l$) take place when computing the partial Fourier coefficients. Note that in (5), the innermost summation $\sum_j$ is moved to the outermost position, and the term $-2(m - \mu)(l - q/2)/N$ is factorized into two independent terms, $(m - \mu)/p$ and $1 - 2l/q$. Interchanging the order of summations and factorizing the term result in a significant computational benefit; we elucidate what operator we should utilize for each summation and how we can save the arithmetic costs from it. As we will see, the innermost sum over $l$ corresponds to a matrix multiplication, the second sum over $k$ can be viewed as a set of DFTs, and the outermost sum over $j$ is an inner product. For the first sum, let $A = (a_{k,l}) = a_{qk+l}$ and $B = (b_{l,j}) = \omega_N^{\mu(l-q/2)} w_{\epsilon,r,j}(1 - 2l/q)^j$, so that (5) can be written as follows:

$$\omega_{2p}^m \sum_{j\in[r]} ((m - \mu)/p)^j \sum_{k\in[p]} \omega_p^{mk} \sum_{l\in[q]} a_{k,l} b_{l,j}.$$

**Algorithm 1:** Configuration phase of PFT

```
input : Input size $N$, output descriptors $M$ and $\mu$, divisor $p$,
        and tolerance $\epsilon$
output: Matrix $B$, divisor $p$, and numbers of rows and
        columns, $q$ and $r$
1 $q \leftarrow N/p$
2 $r \leftarrow \min\{r \in \mathbb{N} : \xi(\epsilon, r) \geq M/p\}$ // $\xi(\epsilon, r)$: precomputed
3 $B[l, j] \leftarrow \omega_N^{\mu(l-q/2)} w_{\epsilon,r,j}(1 - 2l/q)^j$ for $(l, j) \in [q] \times [r]$
                                              // $w_{\epsilon,r,j}$: precomputed
```

**Algorithm 2:** Computation phase of PFT

```
input : Vector $\boldsymbol{a}$ of size $N$, output descriptors $M$ and $\mu$, and
        configuration results $B, p, q, r$
output: Vector $\mathcal{E}(\hat{\boldsymbol{a}})$ of estimated Fourier coefficients of $\boldsymbol{a}$
        for $[\mu - M, \mu + M]$
1 $A[k, l] \leftarrow a_{qk+l}$ for $k \in [p]$ and $l \in [q]$
2 $C \leftarrow A \times B$
3 for $j \in [r]$ do
4     $\hat{C}[\cdot, j] \leftarrow$ FFT($C[\cdot, j]$)   // FFT of $j^{th}$ column of $C$
5 end
6 for $m \in [\mu - M, \mu + M]$ do
7     $\mathcal{E}(\hat{\boldsymbol{a}})[m] \leftarrow \omega_{2p}^m \sum_{j=0}^{r-1}((m - \mu)/p)^j \cdot \hat{C}[m\%p, j]$
8 end
```

Note that the matrix $B$ is data-independent (not dependent on $\boldsymbol{a}$), and thus can be precomputed. Indeed, we have already seen that $\{w_{\epsilon,r,j}\}_{j\in[r]}$ can be precomputed. The other factors $\omega_N^{\mu(l-q/2)}$ and $(1 - 2l/q)^j$ composing the elements of $B$ can also be precomputed if $(N, M, \mu, p, \epsilon)$ is known in advance. Thus, as long as the setting $(N, M, \mu, p, \epsilon)$ is unchanged, we can reuse the matrix $B$ for any input data $\boldsymbol{a}$ once the configuration phase of PFT is completed (Algorithm 1). We shall denote the multiplication $A \times B$ as $C = (c_{k,j})$:

$$\omega_{2p}^m \sum_{j\in[r]} ((m - \mu)/p)^j \sum_{k\in[p]} c_{k,j}\, \omega_p^{mk}. \tag{6}$$

For each $j \in [r]$, the summation $\hat{c}_{m,j} = \sum_{k\in[p]} c_{k,j}\, \omega_p^{mk}$ is a DFT of size $p$. We perform FFT $r$ times for this computation, which yields the following estimation of $\hat{a}_m$ for $m \in R_{\mu,M}$:

$$\omega_{2p}^m \sum_{j\in[r]} ((m - \mu)/p)^j\, \hat{c}_{m,j}. \tag{7}$$

Note that $\hat{c}_{m,j}$ is a periodic function of period $p$ with respect to $m$, so we use the coefficient at $m$ modulo $p$ if $m < 0$ or $m \geq p$. Thus, the $m^{th}$ Fourier coefficient of $\boldsymbol{a}$ can be estimated by the inner product of $((m-\mu)/p)^j$ and $\hat{c}_{m,j}$ with respect to $j$, followed by a multiplication with the extra twiddle factor $\omega_{2p}^m$ (we also precompute $((m-\mu)/p)^j$ and $\omega_{2p}^m$). The full computation is outlined in Algorithm 2. By these summation techniques, the arithmetic complexity is reduced to $O(N + M \log M)$ from naïve $O(MN)$, as described in Section 3.4.

## 3.4 Theoretical Analysis

We present theoretical analysis on the time complexity of PFT and its approximation bound.

*3.4.1 Time Complexity.* We analyze the time complexity of PFT. Theorem 3 shows that the time cost $T(N, M, \epsilon)$ of PFT is $O((N + M \log M) \log(1/\epsilon))$, where $N$ and $M$ are input and output size descriptors, respectively, and $\epsilon$ is a given tolerance. For finding a small number $M$ of coefficients, where $M \ll N$, PFT is much faster than a typical FFT which takes $O(N \log N)$. Note that the theorem presumes that all prime factors of $N$ have a fixed upper bound. Yet, in practice, this necessity is not a big concern because one can readily control the input size with basic techniques such as zero-padding or re-sampling. Moreover, we empirically find that even if $N$ has a large prime factor, PFT still shows a promising performance (see Section 4.2). We first derive an asymptotic equation between $r$ and $\epsilon$ in Lemma 2, and use it for the proof of Theorem 3.

Lemma 2. *Let $c_1 \leq \xi(\epsilon, r) \leq c_2$ for some constants $c_1, c_2 > 0$. Then we have the asymptotic equation, $r = O(\log(1/\epsilon))$.* □

Proof. See Supplement A.2. □

In Theorem 3, note that a positive integer is called $b$-**smooth** if none of its prime factors is greater than $b$. For example, the 2-smooth integers are equivalent to the powers of 2.

Theorem 3. *Fix an integer $b \geq 2$. If $N$ is $b$-smooth, then the time complexity $T(N, M, \epsilon)$ of PFT has an asymptotic upper bound $O((N + M \log M) \log(1/\epsilon))$.* □

Proof. We first claim that the following statement holds: *let $b \geq 2$; if $N$ is $b$-smooth and $M \leq N$ is a positive integer, then there exists a positive divisor $p$ of $N$ satisfying $M/\sqrt{b} \leq p < \sqrt{b}M$.* Indeed, suppose that none of $N$'s divisors belongs to $[M/\sqrt{b}, \sqrt{b}M)$. Let $1 = p_1 < p_2 < \cdots < p_d = N$ be the enumeration of all positive divisors of $N$ in increasing order. It is clear that $p_1 < \sqrt{b}M$ and $M/\sqrt{b} < p_d$ since $b \geq 2$ and $1 \leq M \leq N$. Then, there exists an $i \in \{1, 2, \cdots, d - 1\}$ so that $p_i < M/\sqrt{b}$ and $p_{i+1} \geq \sqrt{b}M$. Since $N$ is $b$-smooth and $p_i < N$, at least one of $2p_i, 3p_i, \cdots, bp_i$ must be a divisor of $N$. However, this is a contradiction because we have $p_{i+1}/p_i > (\sqrt{b}M)(M/\sqrt{b})^{-1} = b$, so none of $2p_i, 3p_i, \cdots, bp_i$ can be a divisor of $N$, which completes the proof.

Exploiting the above property, we manage to reduce the time complexity of PFT to a functional form dependent of only $N$, $M$ and $\epsilon$. We follow the convention in counting FFT operations, assuming that all data-independent elements such as configuration results $B, p, q, r$ and twiddle factors are precomputed, and thus not included in the run-time cost. We begin with the construction of the matrix $A$. For this, we merely interpret $\boldsymbol{a}$ as an array representation for $A$ of size $p \times q = N$ (line 1 in Algorithm 2). Also, recall that the matrix $B$ can be precomputed as described in Section 3.3. For the two matrices $A$ of size $p \times q$ and $B$ of size $q \times r$, standard matrix multiplication algorithm has running time of $O(pqr) = O(r \cdot N)$ (line 2 in Algorithm 2). Next, the expression (6) contains $r$ DFTs of size $p$, namely $\hat{c}_{m,j} = \sum_{k\in[p]} c_{k,j} \cdot \omega_p^{mk}$ for each $j \in [r]$. We use FFT $r$ times for the computation, then it is easy to see that the time cost is given by $O(r \cdot p \log p)$ (lines 3-5 in Algorithm 2). Finally, there are $2M + 1$ coefficients to be calculated in (7), each requiring

$O(r)$ operations, giving an upper bound $O(r \cdot M)$ for the running time (lines 6-8 in Algorithm 2). Combining the three upper bounds $O(r \cdot N)$, $O(r \cdot p \log p)$, and $O(r \cdot M)$, we formally express the time complexity $T(N, M, \epsilon)$ of PFT as follows:

$$T(N, M, \epsilon) = O(r \cdot (N + p \log p + M)).$$

Note that $r$ is dependent of $\epsilon$ and $M/p$ by its definition (line 2 in Algorithm 1). By the preceding argument, we can always find a divisor $p$ of $N$ such that $M/\sqrt{b} \leq p < \sqrt{b}M$, implying that $M/p$ is tightly bounded. Then, it follows that $p = \Theta(M)$ and that $r = O(\log(1/\epsilon))$ by Lemma 2. This leads to the following asymptotic upper bound with respect to $N$, $M$ and $\epsilon$:

$$T(N, M, \epsilon) = O((N + M \log M) \log(1/\epsilon)),$$

hence the proof. □

*3.4.2 Approximation Bound.* We now give a theoretical approximation bound of the estimation via the polynomial $\mathcal{P}$. We denote the estimated Fourier coefficient of $\boldsymbol{a}$ as $\mathcal{E}(\hat{\boldsymbol{a}})$. Theorem 4 states that the approximation bound over the target range is data-dependent of the total weight $\|\boldsymbol{a}\|_1$ of the original vector and the given tolerance $\epsilon$, where $\|\cdot\|_1$ denotes the $\ell_1$ norm. Recall that $\epsilon$ controls the number $r$ of approximating terms (Lemma 2) and thus the error bound $\|\hat{\boldsymbol{a}} - \mathcal{E}(\hat{\boldsymbol{a}})\|_{R_{\mu,M}}$. This indicates that the Fourier coefficients can be evaluated to arbitrary numerical precision with PFT by setting different $\epsilon$ as necessary. Note that the trade-off between accuracy and running time is presented in Theorem 3.

Theorem 4. *Given a tolerance $\epsilon > 0$, the following inequality holds for PFT:*

$$\|\hat{\boldsymbol{a}} - \mathcal{E}(\hat{\boldsymbol{a}})\|_{R_{\mu,M}} \leq \|\boldsymbol{a}\|_1 \cdot \epsilon.$$ □

Proof. See Supplement A.3. □

## 4 EXPERIMENTS

Through experiments, the following questions should be answered:

**Q1 Run-time cost (Section 4.2).** How quickly does PFT compute a part of Fourier coefficients compared to other competitors without sacrificing accuracy?

**Q2 Effect of hyperparameter $p$ (Section 4.3).** How do the different choices of divisor $p$ of input size $N$ affect the overall performance of PFT?

**Q3 Effect of different precision (Section 4.4).** How do the different precision settings affect the running time of PFT?

**Q4 Anomaly detection (Section 4.5).** How well does PFT work for a practical application using FFT (anomaly detection)? What are the discoveries of analyzing real world data with PFT?

### 4.1 Experimental Setup

**Machine.** A machine equipped with Intel Core i7-6700HQ @ 2.60GHz and 8GB of RAM is used.

**Datasets.** We use both synthetic and real-world datasets summarized in Table 2. For $n = 12, \cdots, 22$, $S_n$ is a set of 1000 vectors of length $2^n$ whose elements are random real numbers between 0 and 1. Urban Sound contains 4347 sound recordings in urban environment, and Air Condition is composed of 29 time-series vectors of air condition information (e.g., temperature and humidity). Stock is a new public data we release; it consists of the daily historical stock prices of FANG, the four American technology companies Facebook, Amazon, Netflix, and Google. We collected closing prices adjusted for stock splits, from 2017-01-03 to 2021-01-08.

**Table 2: Summary of datasets.**

| Dataset | Type | # of Time Series | Length |
|---|---|---:|---:|
| $\{S_n\}_{n=12}^{22}$ | Synthetic | 1000 | $2^n$ |
| Urban Sound[1] | Real-world | 4347 | 32000 |
| Air Condition[2] | Real-world | 29 | 19735 |
| Stock[3] | Real-world | 4 | 1012 |

**Competitors.** We compare PFT with two state-of-the-art FFT algorithms FFTW and MKL, as well as Pruned FFTW. All of them are implemented in C++.

(1) **FFTW**: FFTW[4] [7] is one of the fastest public implementation for FFT, offering a hardware-specific optimization. We use the optimized version of FFTW 3.3.5, and do not include the preprocessing for the optimization as the run-time cost.
(2) **MKL**: Intel Math Kernel Library[5] (MKL) is a library of optimized math routines including FFT, and often shows a better running time result than the FFTW. All the experiments are conducted with an Intel processor for the best performance.
(3) **Pruned FFT**: Pruned FFT[6] [1, 16, 19, 29, 31] is a pruned version of FFTW designed for fast computation of a subset of the outputs. The algorithm uses the optimized FFTW as a subroutine.
(4) **PFT (proposed)**: we use MKL BLAS routines for the matrix multiplication, MKL DFTI functions for the batch FFT computation, and Intel Integrated Performance Primitives (IPP) library for the post-processing steps such as inner product and element-wise multiplication.

We have also conducted experiments for Goertzel algorithm [3, 8]. However, we opt not to include the results in the paper because the method shows an extremely poor performance, with up to $10^5\times$ slower running time compared to other competitors.

**Measure.** In all experiments, we use single-precision floating-point format, and the parameters $p$ and $\epsilon$ are chosen so that the relative $\ell_2$ error is strictly less than $10^{-6}$, which ensures that the overall estimated coefficients have at least 6 significant figures. Explicitly,

$$\text{Relative } \ell_2 \text{ Error} = \sqrt{\frac{\sum_{m \in \mathcal{R}} |\hat{a}_m - \mathcal{E}(\hat{a})_m|^2}{\sum_{m \in \mathcal{R}} |\hat{a}_m|^2}} < 10^{-6},$$

where $\hat{\boldsymbol{a}}$ is the actual coefficient, $\mathcal{E}(\hat{\boldsymbol{a}})$ is the estimation of $\hat{\boldsymbol{a}}$, and $\mathcal{R}$ is the target range. Section 4.4 is an exception, where we investigate different settings, varying the precision to $10^{-4}$ or $10^{-2}$.

### 4.2 Run-Time Cost

We evaluate the running time of PFT on synthetic and real-world datasets, varying input and output sizes.

[1]https://urbansounddataset.weebly.com/urbansound8k.html
[2]https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction
[3]https://github.com/snudatalab/PFT
[4]http://www.fftw.org/index.html
[5]https://software.intel.com/mkl
[6]http://www.fftw.org/pruned.html

*4.2.1 Run-Time Cost on Synthetic Data.* We generate 1000 random synthetic vectors of size $2^n$ and evaluate the average running time of PFT over 100 runs for all the vectors with different settings.

**Running time vs. input size.** We fix the target range to $R_{0,2^9}$ and evaluate the average running time of PFT vs. input sizes $N$: $2^{12}$, $2^{13}, \cdots, 2^{22}$. Figure 1(a) shows how the four competitive algorithms scale with varying input size, wherein PFT outperforms the others if the output size is sufficiently smaller ($< 10\%$) than the input size. Consequently, PFT achieves up to 19× speedup compared to its competitors. Due to the overhead of the $O(N)$ pre- and $O(M)$ post-processing steps, PFT runs slower than FFT when $M$ is close to $N$ so the time complexity tends to $O(N + N \log N)$ in the case.

**Running time vs. output size.** We fix the input size to $N = 2^{22}$ and evaluate the average running time of PFT vs. target ranges $R_{0,2^9}, R_{0,2^{10}}, \cdots, R_{0,2^{19}}$. The result is illustrated as a running time vs. output size plot (recall that $|R_{0,M}| \simeq 2M$) in Figure 1(b). Note that PFT significantly extends the range of output sizes for which partial Fourier transform becomes practical. On the other hand, the running times of FFTW and MKL do not benefit from the information of output size. We also observe that Pruned FFT shows only a modest improvement compared to the full FFTs.

*4.2.2 Run-Time Cost on Real-World Data.* When it comes to real-world data, it is not generally the case that the size of an input vector is a power of 2. Notably, PFT still shows a promising performance regardless of the fact that the input size is not a power of 2 or even it has a large prime factor: a strong indication that our proposed technique is robust for many different applications in real-world.

**Urban Sound.** Urban Sound dataset is composed of 4347 sound recordings of length $N = 32000 = 2^8 \times 5^3$. We evaluate the running time of PFT vs. output size ranging from 100 to 6400. Figure 2(a) shows the result, where each point is the average over 100 runs for all 4347 instances. We observe that PFT outperforms the competitors if the output size is small enough compared to the input size.

**Air Condition.** Air Condition dataset contains 29 time-series vectors of $N = 19735 = 5 \times 3947$. Note that $N$ has only two non-trivial divisors, namely 5 and 3947, forcing one to choose $p = 3947$ in any practical settings; if $p = 5$, the ratio $M/p$ turns out to be *too large*, which results in a poor performance (see Section 4.3 for more discussion of the optimal choice of $p$). The running time of PFT vs. output size ranging from 125 to 16000 is evaluated in Figure 2(b), where each point is the average over 1000 runs for all 29 instances (Pruned FFT is not included since it consistently runs slower than all the other competitors). It is noteworthy that PFT still outperforms its competitors even in such pathological examples, implying the robustness of our algorithm for various real-world situations.

## 4.3 Effect of Hyperparameter $p$

To investigate the effect of different choices of $p$, we fix $N = 2^{22}$ and vary the ratio $M/p$ from 1/32 to 4 for target ranges $R_{0,2^9}, R_{0,2^{10}}, \cdots, R_{0,2^{18}}$. Table 3 shows the results, where the bold highlights the best choice of $M/p$ for each $M$, and the missing entries are due to worse performance than FFT. One crucial observation is as follows: with the increase of output size, the best choice of $M/p$ also increases or, equivalently, the optimal value of $p$ tends to be stable. Intuitively, this is the consequence of "balancing" the three summation steps (Section 3.3): when $M \ll N$, the most computationally expensive operation is the matrix multiplication with $O(rN)$ time cost, and thus, $M/p$ should be small so that the $r$ decreases, despite a sacrifice in the batch FFT step requiring $O(rp \log p)$ operations. As the $M$ becomes larger, however, more concern is needed regarding the batch FFT and post-processing steps, so the parameter $p$ should not change rapidly. This implies the possibility that the optimal value of $p$ can be algorithmically auto-selected given a setting $(N, M, \mu, \epsilon)$, which we leave as a future work.

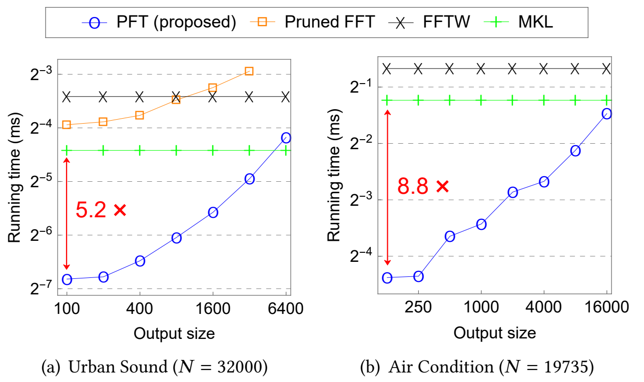


**Figure 2: Running time vs. output size for (a) *Urban Sound* and (b) *Air Condition*. We set the precision of all the methods the same, by making the relative error strictly less than $10^{-6}$. PFT outperforms the competitors regardless of the fact that the input size is not a power of 2 ($32000 = 2^8 \times 5^3$) or even it has a large prime factor ($19735 = 5 \times 3947$). Pruned FFT is not included in (b) since it consistently runs slower than FFTW.**

**Table 3: Average running time (ms) of PFT for $N = 2^{22}$ with different settings of $M$ and $M/p$. With the increase of output size, the best choice of $M/p$ also increases or, equivalently, the optimal value of $p$ tends to remain stable.**

| $M$ | $M/p$ | | | | | |
|---|---|---|---|---|---|---|
| | 1/32 | 1/8 | 1/2 | 1 | 2 | 4 |
| $2^9$ | **1.273** | 2.674 | 2.627 | 2.677 | 4.005 | 4.090 |
| $2^{10}$ | **1.394** | 1.608 | 3.738 | 2.685 | 2.723 | 4.295 |
| $2^{11}$ | 1.634 | **1.332** | 2.717 | 3.805 | 2.731 | 2.986 |
| $2^{12}$ | 2.303 | **1.491** | 1.678 | 2.808 | 3.533 | 2.983 |
| $2^{13}$ | 5.659 | 1.860 | **1.526** | 1.687 | 2.878 | 4.108 |
| $2^{14}$ | 14.121 | 3.020 | 1.881 | **1.692** | 1.949 | 3.275 |
| $2^{15}$ | - | 7.711 | 2.707 | 2.164 | **1.940** | 2.365 |
| $2^{16}$ | - | - | 5.740 | 3.530 | **2.821** | 2.929 |
| $2^{17}$ | - | - | 14.715 | 7.749 | 5.556 | **5.411** |
| $2^{18}$ | - | - | - | - | 12.534 | **11.924** |

## 4.4 Effect of Different Precision

Recall that PFT provides an option to set an arbitrary numerical precision (Theorem 4). In Section 4.2, we fixed the precision of all the methods to $< 10^{-6}$, where PFT showed the fastest running time.

The next questions are as follows: *is it possible to reduce the running time of PFT even further if we relax the precision requirement? What is the trade-off between accuracy and running time?* To investigate them, we fix $N = 2^{22}$ and change the precision goal from $10^{-6}$

**Table 4: Average running time (ms) of PFT for $N = 2^{22}$ with different precision settings. Note that the running time reduces by up to 17% or 27%, when relaxing the precision requirements. This trade-off is useful especially when the fast evaluation is of utmost importance.**

| $M$ | Precision | | |
|---|---|---|---|
| | $10^{-6}$ | $10^{-4}$ | $10^{-2}$ |
| $2^9$ | 1.273 | 1.249 (.98) | 1.238 (.97) |
| $2^{10}$ | 1.295 | 1.278 (.99) | 1.244 (.96) |
| $2^{11}$ | 1.332 | 1.293 (.97) | 1.251 (.94) |
| $2^{12}$ | 1.491 | 1.329 (.89) | 1.277 (.86) |
| $2^{13}$ | 1.526 | 1.400 (.92) | 1.343 (.88) |
| $2^{14}$ | 1.692 | 1.607 (.95) | 1.512 (.89) |
| $2^{15}$ | 1.940 | 1.872 (.96) | 1.740 (.90) |
| $2^{16}$ | 2.821 | 2.469 (.88) | 2.297 (.81) |
| $2^{17}$ | 5.411 | 4.590 (.85) | 4.058 (.75) |
| $2^{18}$ | 11.924 | 9.927 (.83) | 8.733 (.73) |

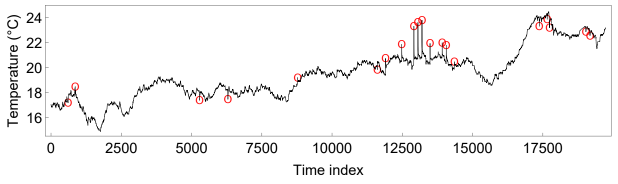
to $10^{-4}$ or $10^{-2}$ for target ranges $R_{0,2^9}, \cdots, R_{0,2^{18}}$. Table 4 shows the results, where the parenthesized number is the ratio of running times of each setting to $10^{-6}$. Note that they are reduced by up to 17% or 27% when the precision goal is set to $10^{-4}$ or $10^{-2}$, respectively. This observation indicates that one may readily benefit from the trade-off, especially when the fast evaluation is of utmost importance albeit with a slight sacrifice in accuracy.

## 4.5 Anomaly Detection

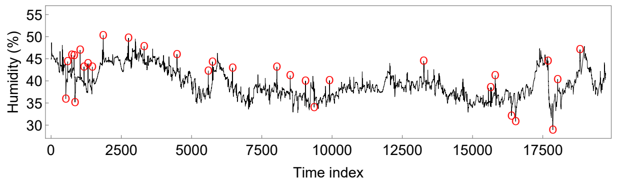
We present application of PFT for anomaly detection. Here is one simple but fundamental principle: *replace the "do FFT and discard unused coefficients" procedure with "just do PFT."* Considering the anomaly detection method proposed in [23], where one first performs FFT and then inverse FFT with only a few low-frequency coefficients to obtain an estimated fitted curve, we can directly apply the principle to the method. We first verify the accuracy of anomaly detection after replacing the standard FFT with PFT, and show the efficacy of PFT by presenting interpretations of anomalous points in time-series data.

*4.5.1 Accuracy.* To validate the accuracy of anomaly detection with PFT, we use time-series vectors with various patterns from Air Condition dataset, and set the target range as $R_{0,125}$ ($\simeq$ 250 low-frequency coefficients). Note that, in this setting, PFT results in $\sim 8\times$ speedup compared to the conventional FFT (see Figure 2(b)). We then construct an estimated fitted curve with the coefficients, and compute the absolute difference between the original and fitted curves to obtain the top-k largest points in magnitude. The top-k anomalous points detected from the time series are presented in Figure 3. As a result, all the anomalies found by PFT exactly coincide with those found by FFT. Therefore, replacing FFT with PFT allows both fast and accurate anomaly detection on time series, regardless of different patterns of the data.

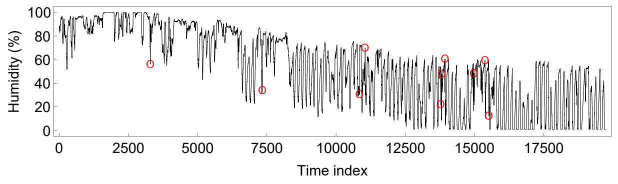
*4.5.2 Discovery.* We have shown that PFT successfully detects the anomalies or local outliers in a given time-series vector. This in turn raises intriguing questions: *what is the interpretation of the anomalies, and how is an anomalous point related to a real-world event?* To answer these questions, we present a set of discoveries of analyzing historical stock price data by PFT. With Stock dataset containing four time-series vectors of stock prices for Facebook, Amazon, Netflix, and Google, we conduct top-5 anomaly detection tasks using PFT, and interpret each anomalous point by matching it to a real-world incident that occurred in the corresponding period. Figure 4 shows the anomalies found by PFT (they are also the same as those by FFT). We observe that most of the anomalies are distributed in the period after the outbreak of COVID-19, mainly because of the increasing uncertainty. There also exist particular factors by which a certain company is more affected. For example, the US-China trade war had the greatest impact on Amazon among the four companies, and the US election had affected Facebook and Google the more. These observations offer an interpretation of the anomaly detection, validating the efficacy of our proposed method.



(a) Pattern 1: relatively smooth time series



(b) Pattern 2: moderately oscillatory time series

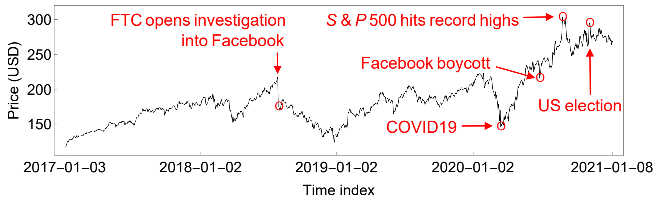


(c) Pattern 3: highly oscillatory time series

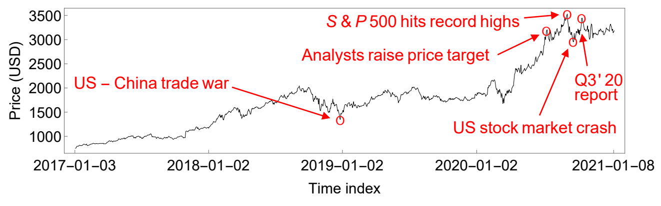
**Figure 3: Top-k anomaly detection using PFT on time-series vectors with various patterns. Note that all the anomalous points (red circles) found by PFT exactly coincide with those found by standard FFT, and that PFT results in $\sim 8\times$ speedup compared to FFT, regardless of the different patterns of data. This demonstrates the accuracy and robustness of PFT on real-world anomaly detection.**

## 5 CONCLUSIONS

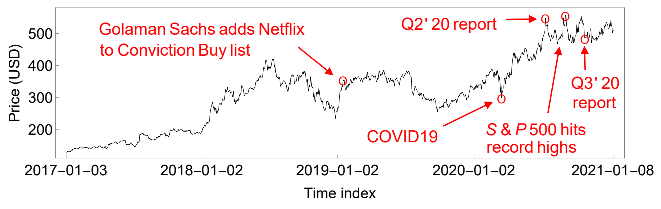
We propose PFT (Partial Fourier Transform), an efficient algorithm for computing a part of Fourier coefficients. PFT approximates some of twiddle factors with relatively small oscillations using polynomials, reducing the computational complexity of DFT due to the mixture of many twiddle factors. Experimental results show that PFT outperforms the state-of-the-art FFTs as well as pruned FFT, with an order of magnitude of speedup without accuracy loss, significantly extending the range of applications where partial Fourier



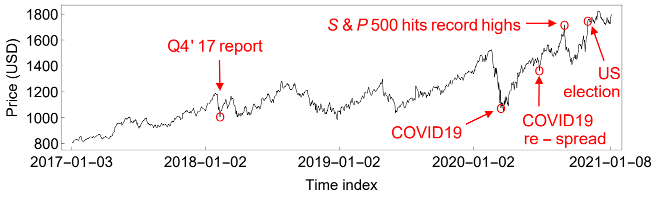
(a) Facebook



(b) Amazon



(c) Netflix



(d) Google

**Figure 4: Top-5 anomaly detection using PFT on time series for (a) Facebook, (b) Amazon, (c) Netflix, and (d) Google. We find that each anomalous point (red circle) is closely related to a real-world event during the corresponding period and thus easily interpretable. These results show the efficacy of our proposed method on anomaly detection.**

transform becomes practical. Moreover, PFT provides an option of trading off running time and error, which is useful especially when the fast evaluation is very important. We also demonstrate the accuracy and efficacy of PFT on real-world anomaly detection, presenting interpretations of anomalies in stock price data. Future works include further optimizing the implementation of PFT.

## ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) funded by MSIT(2019R1A2C2004990). The Institute of Engineering Research and ICT at Seoul National University provided research facilities for this work. U Kang is the corresponding author.

## REFERENCES

[1] Nir Ailon and Edo Liberty. 2009. Fast dimension reduction using Rademacher series on dual BCH codes. *Discrete & Computational Geometry* 42, 4 (2009), 615.

[2] C Boncelet. 1986. A rearranged DFT algorithm requiring N 2/6 multiplications. *IEEE Trans. Acoust. Speech Signal Process.* 34, 6 (1986).

[3] C Sidney Burrus and TW Parks. 1985. *DFT/FFT and Convolution Algorithms.* Citeseer.

[4] Neal L Carothers. 1998. A short course on approximation theory. (1998).

[5] James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation* 19, 90 (1965).

[6] W. Fraser. 1965. A Survey of Methods of Computing Minimax and Near-Minimax Polynomial Approximations for Functions of a Single Independent Variable. *J. ACM* 12, 3 (1965), 295–314.

[7] M. Frigo and S. G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

[8] Gerald Goertzel. 1958. An algorithm for the evaluation of finite trigonometric series. *The American Mathematical Monthly* 65, 1 (1958), 34–35.

[9] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Nearly optimal sparse fourier transform. In *ACM symposium on Theory of computing.*

[10] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. 2012. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms.* SIAM, 1183–1194.

[11] AN Hossen, Ulrich Heute, OV Shentov, and SK Mitra. 1995. Subband DFT—part II: accuracy, complexity and applications. *Signal Processing* 41, 3 (1995), 279–294.

[12] Xiaodi Hou and Liqing Zhang. 2007. Saliency Detection: A Spectral Residual Approach. In *CVPR.* IEEE Computer Society.

[13] Piotr Indyk, Michael Kapralov, and Eric Price. 2014. (Nearly) sample-optimal sparse Fourier transform. In *ACM-SIAM symposium on Discrete algorithms.* SIAM.

[14] Jiarui Jin, Jiarui Qin, Yuchen Fang, Kounianhua Du, Weinan Zhang, Yong Yu, Zheng Zhang, and Alexander J. Smola. 2020. An Efficient Neighborhood-based Interaction Model for Recommendation on Heterogeneous Graph. *CoRR* (2020).

[15] Osman Asif Malik and Stephen Becker. 2018. Low-Rank Tucker Decomposition of Large Tensors Using TensorSketch. In *NeurIPS.* 10117–10127.

[16] J Markel. 1971. FFT pruning. *IEEE transactions on Audio and Electroacoustics* 19, 4 (1971), 305–311.

[17] Michaël Mathieu, Mikael Henaff, and Yann LeCun. 2014. Fast Training of Convolutional Networks through FFTs. In *ICLR.*

[18] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast approximate correlation for massive time-series data. In *SIGMOD.* ACM, 171–182.

[19] K Nagai. 1986. Pruning the decimation-in-time FFT algorithm with frequency shift. *IEEE Trans. Acoust. Speech Signal Process.* 34, 4 (1986), 1008–1010.

[20] Rasmus Pagh. 2013. Compressed matrix multiplication. *ACM Trans. Comput. Theory* 5, 3 (2013), 9:1–9:17.

[21] Ninh Pham and Rasmus Pagh. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *KDD.* ACM, 239–247.

[22] Peng Qi, Juan Cao, Tianyun Yang, Junbo Guo, and Jintao Li. 2019. Exploiting multi-domain visual information for fake news detection. In *ICDM.* IEEE.

[23] Faraz Rasheed, Peter Peng, Reda Alhajj, and Jon G. Rokne. 2009. Fourier Transform Based Spatial Outlier Mining. In *IDEAL*, Vol. 5788. Springer, 317–324.

[24] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *KDD.* ACM, 3009–3017.

[25] Oren Rippel, Jasper Snoek, and Ryan P Adams. 2015. Spectral representations for convolutional neural networks. In *NeurIPS.* 2449–2457.

[26] Tim Schlüter and Stefan Conrad. 2010. An Approach for Automatic Sleep Stage Scoring and Apnea-Hypopnea Detection. In *ICDM.* IEEE, 1007–1012.

[27] OV Shentov, SK Mitra, Ulrich Heute, and AN Hossen. 1995. Subband DFT—Part I: Definition, interpretation and extensions. *Signal Processing* 41, 3 (1995), 261–277.

[28] Sheng Shi, Runkai Yang, and Haihang You. 2017. A new two-dimensional Fourier transform algorithm based on image sparsity. In *ICASSP.* IEEE, 1373–1377.

[29] D Skinner. 1976. Pruning the decimation in-time FFT algorithm. *IEEE Trans. Acoust. Speech Signal Process.* 24, 2 (1976), 193–194.

[30] Georgey S Smirnov and Roman G Smirnov. 1999. Best uniform approximation of complex-valued functions by generalized polynomials having restricted ranges. *Journal of approximation theory* 100, 2 (1999), 284–303.

[31] Henrik V Sorensen and C Sidney Burrus. 1993. Efficient computation of the DFT with only a subset of input or output points. *IEEE transactions on signal processing* 41, 3 (1993), 1184–1200.

[32] Petr Sysel and Pavel Rajmic. 2012. Goertzel algorithm generalized to non-integer multiples of fundamental frequency. *EURASIP Journal on Advances in Signal Processing* 2012, 1 (2012), 56.

[33] Qingsong Wen, Zhe Zhang, Yan Li, and Liang Sun. 2020. Fast RobustSTL: Efficient and Robust Seasonal-Trend Decomposition for Time Series with Complex Patterns. In *KDD.* ACM, 2203–2213.

[34] Liheng Zhang, Charu Aggarwal, and Guo-Jun Qi. 2017. Stock price prediction via discovering multi-frequency trading patterns. In *KDD.* ACM, 2141–2149.

# A SUPPLEMENT

## A.1 Proof of Lemma 1

Proof. Let $Q = \arg\min_{P \in P_\alpha} \|P(x) - e^{uix}\|_{|x-\mu| \le |\xi|}$. We first observe the following equation:

$$\begin{aligned} Q &= \arg\min_{P \in P_\alpha} \|P(x) - e^{uix}\|_{|x-\mu| \le |\xi|} \\ &= \arg\min_{P \in P_\alpha} \|P(x+\mu) - e^{ui(x+\mu)}\|_{|x| \le |\xi|} \\ &= \arg\min_{P \in P_\alpha} \|e^{-ui\mu} \cdot P(x+\mu) - e^{uix}\|_{|x| \le |\xi|}, \end{aligned}$$

where the third equality holds since $|e^{-ui\mu}| = 1$. Recall that the polynomial $\mathcal{P}_{\alpha,\xi,u}$ is defined by $\arg\min_{P \in P_\alpha} \|P(x) - e^{uix}\|_{|x| \le |\xi|}$. If $Q(x) \in P_\alpha$, it is clear that $e^{-ui\mu} \cdot Q(x+\mu) \in P_\alpha$ because translation and non-zero scalar multiplication on a polynomial do not change its degree. Therefore, by the uniqueness of the best approximation [30], we have

$$e^{-ui\mu} \cdot Q(x+\mu) = \mathcal{P}_{\alpha,\xi,u}(x),$$

which yields $Q(x) = e^{ui\mu} \cdot \mathcal{P}_{\alpha,\xi,u}(x-\mu)$, and hence the proof. □

## A.2 Proof of Lemma 2

Proof. Let $\xi(\epsilon, r) = c$ for a constant $c$ between $c_1$ and $c_2$, and consider the best polynomial approximation to $e^{c\pi ix}$ on $|x| \le 1$ (recall Definition 3.2). We shall make use of the Taylor series $e^{c\pi ix} = \sum_{n \ge 0} (c\pi ix)^n/n!$. Note that for a non-negative integer $n$, the $n$-th power of $x$ can be written as follows:

$$x^n = \frac{1}{2^{n-1}}\left(T_n(x) + \binom{n}{1}T_{n-2}(x) + \binom{n}{2}T_{n-4}(x) + \cdots\right),$$

where $T_n(x)$ is the *Chebyshev polynomial* [4] of degree $n$ (for even $n$, the coefficient of $T_0(x)$ is divided by 2). Then, we have

$$e^{c\pi ix} = \sum_{n \ge 0} \frac{(c\pi ix)^n}{n!} = \sum_{n \ge 0} \frac{(c\pi i)^n}{n!} \frac{1}{2^{n-1}} \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{k} T_{n-2k}(x).$$

Dropping the $T_{n-2k}$ terms for $n - 2k \ge r$ gives a polynomial approximation of degree less than $r$, with an error $\epsilon$ at most

$$\sum_{n-2k \ge r} \left| \frac{(c\pi i)^n}{n!} \frac{1}{2^{n-1}} \binom{n}{k} \right| = \sum_{n-2k \ge r} \frac{(c\pi)^n}{n!} \frac{1}{2^{n-1}} \binom{n}{k}.$$

We may rewrite this as

$$\epsilon \le \sum_{n \ge r} \sum_{k \le \lfloor (n-r)/2 \rfloor} \frac{(c\pi)^n}{n!} \frac{1}{2^{n-1}} \binom{n}{k}.$$

Since $\sum_{k=0}^{n} \binom{n}{k} = 2^n$, the summation has the following upper bound:

$$2 \sum_{n \ge r} \frac{(c\pi)^n}{n!}.$$

Suppose that $r > c\pi$, then we have

$$\begin{aligned} \epsilon &\le 2 \sum_{n \ge r} \frac{(c\pi)^n}{n!} \\ &\le 2\left(\frac{(c\pi)^r}{r!} + \frac{(c\pi)^{r+1}}{r! \cdot r} + \frac{(c\pi)^{r+2}}{r! \cdot r^2} + \cdots\right) \\ &= 2\frac{(c\pi)^r}{r!} \frac{1}{1 - (c\pi/r)} \\ &\le c' \frac{(c\pi)^r}{r!}, \end{aligned}$$

where $c'$ is a constant. Note that the similar argument holds for all $c_1 \le c \le c_2$ whenever $r > c_2\pi$. This implies that there exists a constant $C > 0$ satisfying $\epsilon \le C/2^r$ for all sufficiently large $r$. It follows that $2^r \le C/\epsilon$, and thus

$$r = O(\log(1/\epsilon)),$$

which completes the proof. □

## A.3 Proof of Theorem 4

Proof. Let $v = l - q/2$ and $\mathcal{P} = \mathcal{P}_{r-1,\xi(\epsilon,r),\pi}$. By the estimation in (5), the following holds:

$$\begin{aligned} &\|\hat{\boldsymbol{a}} - \mathcal{E}(\hat{\boldsymbol{a}})\|_{R_{\mu,M}} \\ &= \|\sum_{k,l} a_{qk+l}\big(\omega_N^{vm} - \omega_N^{v\mu}\mathcal{P}(-2v(m-\mu)/N)\big)\omega_p^{mk}\omega_{2p}^{m}\|_{m \in R_{\mu,M}} \\ &\le \sum_{k,l} \|a_{qk+l}\big(\omega_N^{vm} - \omega_N^{v\mu}\mathcal{P}(-2v(m-\mu)/N)\big)\omega_p^{mk}\omega_{2p}^{m}\|_{m \in R_{\mu,M}} \\ &= \sum_{k,l} |a_{qk+l}| \cdot \|\omega_N^{v(m-\mu)} - \mathcal{P}(-2v(m-\mu)/N)\|_{m \in R_{\mu,M}}, \end{aligned}$$

since we have $|\omega_p^{mk}| = |\omega_{2p}^{m}| = |\omega_N^{v\mu}| = 1$. If $l$ ranges from 0 to $q-1$, then $|2v/N| \le 2(q/2)/N = 1/p$, and thus, $M|2v/N| \le M/p \le \xi(\epsilon, r)$. We extend the domain of the function $\omega_N^{v(m-\mu)} - \mathcal{P}(-2v(m-\mu)/N)$ from $m \in R_{\mu,M} = [\mu - M, \mu + M] \cap \mathbb{Z}$ to $x \in [\mu - M, \mu + M]$ (note that extending domain never decreases the uniform norm), and replace $-2v(x-\mu)/N$ with $x'$, from which it follows that

$$\begin{aligned} &\|\hat{\boldsymbol{a}} - \mathcal{E}(\hat{\boldsymbol{a}})\|_{R_{\mu,M}} \\ &\le \sum_{k,l} |a_{qk+l}| \cdot \|e^{-2\pi iv(x-\mu)/N} - \mathcal{P}(-2v(x-\mu)/N)\|_{|x-\mu| \le M} \\ &= \sum_{k,l} |a_{qk+l}| \cdot \|e^{\pi ix'} - \mathcal{P}(x')\|_{|x'| \le M|2v/N|} \\ &\le \sum_{k,l} |a_{qk+l}| \cdot \|e^{\pi ix'} - \mathcal{P}(x')\|_{|x'| \le \xi(\epsilon,r)} \\ &\le \sum_{k,l} |a_{qk+l}| \cdot \epsilon \\ &= \|\boldsymbol{a}\|_1 \cdot \epsilon, \end{aligned}$$

where the second inequality holds since $M|2v/N| \le \xi(\epsilon, r)$. This completes the proof. □