Relaxed Functional Bootstrapping:
A New Perspective on BGV and BFV Bootstrapping

Zeyu Liu
zeyu.liu@yale.edu
Yale University

Yunhao Wang
yunhao.wang@yale.edu
Yale University

November 24, 2024

Abstract

BGV and BFV are among the most widely used fully homomorphic encryption (FHE) schemes, supporting evaluations over a finite field. To evaluate a circuit with arbitrary depth, bootstrapping is needed. However, despite the recent progress, bootstrapping of BGV/BFV still remains relatively impractical, compared to other FHE schemes.

In this work, we inspect the BGV/BFV bootstrapping procedure from a different angle. We provide a generalized bootstrapping definition that relaxes the correctness requirement of regular bootstrapping, allowing constructions that support only certain kinds of circuits with arbitrary depth. In addition, our definition captures a form of functional bootstrapping. In other words, the output encrypts a function evaluation of the input instead of the input itself.

Under this new definition, we provide a bootstrapping procedure supporting different types of functions. Our construction is 1-2 orders of magnitude faster than the state-of-the-art BGV/BFV bootstrapping algorithms, depending on the evaluated function.

Of independent interest, we show that our technique can be used to improve the batched FHEW/TFHE bootstrapping construction introduced by Liu and Wang (Asiacrypt 2023). Our optimization provides a speed-up of 6x in latency and 3x in throughput for batched binary gate bootstrapping and a plaintext-space-dependent speed-up for batched functional bootstrapping with plaintext space smaller than $\mathbb{Z}_{512}$.

Contents

1 Introduction

Fully Homomorphic Encryption (FHE) allows one to securely compute over encrypted data without the knowledge of the secret key or interaction with the owner of the data, thus resulting in a very strong primitive. FHE was first realized by Gentry in the groundbreaking work [27]. Since then, there have been lots of works trying to improve the efficiency of FHE [9, 8, 6, 22, 7, 29, 21, 18, 17].

These works follow a similar paradigm as Gentry's original work: a ciphertext contains some initial noise and each operation (e.g., multiplications and additions) introduces some additional noise; the initial parameter provides some noise budget, and if the noise budget is used up when carrying out operations (i.e., the noise has grown to be close to some threshold), computations cannot be continued. This is thus called the Leveled Homomorphic Encryption (LHE). To make LHE indeed FHE, one needs an additional operation: bootstrapping.

Essentially, bootstrapping takes a ciphertext with a relatively large error (i.e., only a small amount of noise budget left) and outputs a new ciphertext encrypting the same plaintext with a relatively small error (i.e., a large amount of noise budget left). With bootstrapping, one can evaluate circuits of arbitrary depths. Bootstrapping itself is very costly and a lot of work has been done to improve the efficiency of bootstrapping (e.g., [21, 18, 54, 16, 13]).

Among all the FHE schemes, one critical line of work is BGV/BFV [7, 6, 22], which has a very important property called "Same Instruction Multiple Data" (SIMD), i.e., one BFV ciphertext encrypts a vector of $D \gg 1$ $\mathbb{Z}_t$ elements (in D slots). Then, any operation over a ciphertext is done over all these D $\mathbb{Z}_t$ elements (element-wise). The state-of-the-art framework for BGV/BFV bootstrapping was first introduced in [33] and later improved in [14, 26, 25, 59]. While the most recent work [59] has asymptotically achieved a very efficient construction (requiring only a logarithmic number of ciphertext multiplications), the practicality is still relatively limited. Concretely, to bootstrap for a single element (amortized over D slots) in $\mathbb{Z}_{257}$, the state-of-the-art BFV bootstrapping construction takes $\sim$0.17 second [59]. The main issue is that to design a suitable bootstrapping scheme for arbitrary circuits, these works make use of some algebraic structure to improve the efficiency and thus do not take full advantage of the SIMD capability of BGV/BFV. Thus, the amortized cost of every slot is still relatively large.

However, in some cases, supporting arbitrary circuits might be an overkill. In some circuits, there might exist some sub-circuit C such that the output is only a subset of the entire plaintext space $\mathbb{Z}_t$, or there might exist some sub-circuit C' such that only a subset of $\mathbb{Z}_t$ is taken as input. In other words, we have $C : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y} \subset \mathbb{Z}_t$, or $C' : \mathcal{X}' \to \mathcal{Y}'$ where $\mathcal{X}' \subset \mathbb{Z}_t$. This means that if the bootstrapping comes after C or before C', only the plaintext values in $\mathcal{Y}$ or $\mathcal{X}'$ need to be bootstrapped, respectively. Conditional branching serves as a great example here: if $x \in [u, v]$, [1]outputs y; if $x \in [u', v']$, outputs y'. This branching can then be modeled as a circuit $f : \mathcal{X}_f \to \mathcal{Y}_f$, where $\mathcal{X}_f := [u, v] \cup [u', v']$ and $\mathcal{Y}_f := \{y, y'\}$. Both $\mathcal{X}_f$ and $\mathcal{Y}_f$ are only a (small) subset of $\mathbb{Z}_t$.

In fact, such circuits are common in many applications, like oblivious permutation [23], PSI with computation [36], secure machine learning [44, 38, 45, 39, 20], and so on. See Section 7 for more examples and detailed discussions. Therefore, the first natural question we ask in this paper is:

Can we achieve better efficiency by relaxing the requirement of supporting arbitrary circuits? In other words, if we only allow bootstrapping input to be a subset of the entire plaintext space, can we do bootstrapping more efficiently?

Another inefficiency of regular bootstrapping comes from the fact that, as a standalone component, the bootstrapping procedure itself does not directly contribute to evaluating the target circuit. Every effort spent on bootstrapping is an extra cost. Therefore, the second question we pose is:

Can we achieve better efficiency by embedding bootstrapping into the circuit? In other words, can we do bootstrapping while evaluating a circuit like C or C' without introducing much overhead?

In this paper, we make solid progress in both directions. Due to space reasons, please see our full version [55] for deferred details and clarifications.

[1] $[u, v]$ denotes $\{u, u + 1, \ldots, v\}$.

1.1 Our Contribution

Definition of generalized functional bootstrapping. We propose a new generalized definition of BFV bootstrapping. The definition captures the most fundamental requirement of bootstrapping (i.e., the output ciphertext has a larger noise budget than the input ciphertext), but also allows some relaxation on the functionality: the output ciphertext does not need to provide correctness for *all* plaintexts, but only a predetermined subset of the plaintext space. In other words, for the input ciphertexts encrypting *invalid plaintexts* (i.e., not in that subset), the construction may output a ciphertext encrypting an arbitrary plaintext, since an expected output is not defined. This relaxation allows us to develop more efficient bootstrapping constructions for valid inputs.

Moreover, the definition captures a form of *functional bootstrapping*, which means that the output encrypts $f(x)$ instead of x, where x is the valid input plaintext and f is some predetermined function. This allows the bootstrapping itself to be embedded into the circuit for better efficiency.

As an auxiliary property, we define *closeness*, which captures how the output behaves when the inputs are invalid. Instead of arbitrary output for invalid input, the algorithm returns the expected output of some valid input points *close* to the invalid input in the plaintext space. This property provides extra flexibility and might be useful in some applications.

Constructions of generalized functional bootstrapping. In addition, we show a general framework for this (relaxed) BFV bootstrapping. While our framework cannot be used to achieve regular bootstrapping, we show that it can be used to efficiently achieve relaxed bootstrapping while evaluating three different types of functions:

- Point functions: the function takes m points and maps them to m points. This type of function is like an arbitrary lookup table, but only the specified m points are valid inputs, where $m \ll t$ for t being the plaintext modulus. The runtime is essentially linear in $c \cdot m$ (where c is some tuneable parameter) instead of t as in regular BFV bootstrapping constructions.

- Range functions: the function takes k ranges and maps them to k points, each range containing m consecutive points [2]. This type of function is more limited, but can be evaluated more efficiently than the point functions: even if there are $m \cdot k$ valid input points, the runtime cost is only $(m+c) \cdot k$ instead of $c \cdot m \cdot k$.

- Unbalanced range functions: the function takes 2 ranges and maps them into 2 different points, where the first range contains m_1 points and the second range contains $m_2 \gg m_1$ points. This is a special case of the range functions. However, with $m_2 \gg m_1$, we construct a bootstrapping scheme running in $m_1 + c + \log(t)$ time, which is much more efficient than the $m_1 + m_2 + 2c$ (the efficiency of applying the algorithm for the general range functions). We also extend this result to functions with $k > 2$ ranges, where the k-th range is larger than all the other ranges combined. In this case, our construction can evaluate it more efficiently than naively evaluating it as normal range functions.

We implement our construction as a C++ library, and show that it is indeed concretely more efficient than regular BFV bootstrapping: the amortized cost is about 1-2 orders of magnitude faster than regular BFV bootstrapping, depending on the function that is evaluated by our functional bootstrapping procedure.

To showcase the practicality of our construction, in Section 7 we specifically demonstrate that our relaxed functional bootstrapping constructions could bring a 20x speedup to oblivious permutation [23] compared to using the bootstrapping from prior works, and also discuss some other applications that could take advantage of our constructions.

Batched LWE ciphertexts bootstrapping. As an independent contribution, we show that our techniques can be applied to improve the batched functional bootstrapping construction for LWE ciphertexts introduced by [54]. Our benchmark shows that for binary-gate batched bootstrapping, our construction is about 3x faster than [54]. Moreover, with optimizations allowing the runtime to scale with the plaintext

[2] Each range may contain a different amount of points, but for simplicity here, we assume they all have m points.

space, our construction greatly brings down the overall bootstrapping runtime for smaller plaintext space. Compared to the uniform runtime for any plaintext space smaller than $\mathbb{Z}_{512}$ in the prior work, our work is more efficient when considering the functional/programmable bootstrapping for plaintext space with 3-8-bit.

1.2 Related Work

1.2.1 BFV Bootstrapping

All the prior works about BFV bootstrapping are regular bootstrapping whose goal is simply to reduce the error (or equivalently increase the noise budget) of the input ciphertexts [6, 22, 33, 14, 26]. This allows one to evaluate circuits with arbitrary depth.

In Table 1, we compare our result with the prior works on regular BFV bootstrapping [33, 14, 26, 25, 59]. Our protocol supports several different types of functions: f_{pts} maps a random set $\mathcal{X}$ to another random set $\mathcal{Y}$, with $|\mathcal{Y}| \leq |\mathcal{X}| \leq \lfloor t/r \rfloor$ (for $\mathbb{Z}_t$ being the plaintext space, and $r = O(\sqrt{h})$ where h is the hamming weight of the BFV secret key); f_{ranges} represents a range-to-point mapping for multiple ranges, while f_{ub} maps one range to some point and the other much larger range [3] to another point (i.e., an unbalanced range mapping). Note that f_{ranges} and f_{ub} both map ranges to points and both require the ranges to be separated by r (for example, if the inputs are composed of k ranges, we need $[u_i - r/2, v_i + r/2] \cap [u_j - r/2, v_j + r/2] = \emptyset$ for all $i \neq j \in [k]$). f_1 and f_2 are two additional types of functions that serve as stepping stones toward our final construction, where f_1 is the identity function on a subset of $\mathbb{Z}_t$ with static intervals r, and f_2 is its generalized version with potentially small static intervals.

The closeness property is a new property we define additional to standard correctness (which does not put any constraint on invalid inputs): for any invalid input, the output still needs to be the outputs of one of the closest ℓ valid input points to that invalid input. This property can be useful in some applications but is not a hard requirement as constructions may take advantage of not having it for better efficiency. See Section 7 for further discussion regarding applications (with ands without closeness requirement).

In Algorithm 7 in Section 5.2.2, we provide an alternative construction to evaluate the type f_{ranges} (i.e., type (2) function of our result). This alternative construction is more efficient when the k-th range is larger than the other $k-1$ ranges combined and provides k-closeness (Definition 3.2). Let the total size of the first $k-1$ ranges be S. The depth is $\log(S + r(k-1)) + \log(t)$; number of scalar multiplications is S; and the number of non-sclar multiplications is $S + \log(t)$. To avoid extra complexity, we do not include it in the table.

From Table 1, we can see that the total cost of our construction is dependent on the functions and thus more fine-grained. For example, for f_{pts}, if $|X| = O(1)$, our construction cost can also be $O(1)$ (as $r = O(\sqrt{h})$ and h is viewed as a constant in prior works [26, Section 7.2]). Note that since function f_1, f_2 can be treated as special cases of f_{pts}, they achieve the same efficiency asymptotically as f_{pts}. On the other hand, although f_{ub} is also a special case of f_{ranges}, f_{ub} is both asymptotically and concretely more efficient when $|v_1 - u_1| \ll |v_2 - u_2|$ (due to symmetry, it also works for functions with $|v_1 - u_1| \gg |v_2 - u_2|$).

Lastly, our construction supports a lot more slots: N compared to N/d. In practice, $d \gg 1$. For example, for the parameters tested in [14], $N/d \approx \sqrt{N}$ (we refer readers to the caption of Table 1 for more detailed parameter definitions). Thus, prior works inherently support a lot fewer slots due to their techniques and can hardly be extended. Thus, our amortized efficiency is much better than prior works.

Method-wise, our construction uses a different idea to reduce costs. [33, 14, 26] focuses on temporarily enlarging the plaintext space to accommodate the partial decryption value. In more detail, when computing $b - \langle \vec{a}, \mathsf{sk} \rangle$ for ciphertext $(\vec{a}, b)$,[4] instead of computing it over $\mathbb{Z}_{t^e}$, they compute it over $\mathbb{Z}_{t^{e'}}$ for some $e' > 1$ satisfying $b - \langle \vec{a}, \mathsf{sk} \rangle \ll t^{e'}$ when evaluated in the integer domain; thus, they obtain $k \cdot t + m + \epsilon < t^{e'}$ for some integer k. To recover m, one of their main steps is then to remove $k \cdot t$. In contrast, our construction directly computes the partial decryption over $\mathbb{Z}_t$, and thus do not need this step.

[3] Can be almost as large as all the other points in $\mathbb{Z}_t$.

[4] Notice that for readability, we use LWE ciphertext as an illustration here, while in the construction, we use RLWE ciphertext instead.

A recent work [59] explores the algebraic structure of the plaintext space, they can reduce the number of non-scalar multiplications from $O(\sqrt{p})$ to $O(\log(p))$ (other works in the table has $O(\sqrt{p})$ non-scalar multiplications instead). Despite their interesting techniques and great asymptotic improvement, it still only supports N/d slots, and concretely, the construction is only about 1.6x faster than the prior constructions [14]. We provide a concrete comparison in Section 6.

1.2.2 Recent concurrent works

Comparison with [41]. A recent concurrent and independent work [41] uses CKKS to bootstrap BFV. With this interesting novel idea, they achieve a concrete amortized runtime comparable to our construction. While they have some advantages over our solution (they support regular BFV bootstrapping and can more easily support larger plaintext modulus), they also have some disadvantages: (1) since their technique focuses on noise removal, it is inherently hard to extend their scheme to support functional bootstrapping; (2) their concrete runtime increases relatively fast when the noise budget after bootstrapping grows; (3) since they rely on CKKS, two sets of evaluation keys are needed for both CKKS and BFV when using BFV with their bootstrapping method; (4) relying on CKKS may introduce extra security concerns as discussed in [50]; however, note that they only use CKKS as an intermediate step and thus this may not be an issue any all; but it may be a concern for some applications and thus may require further investigation. With these different pros and cons, we believe that our scheme and their scheme may be preferred by different applications. We give a concrete comparison in Section 6.

Comparison with [56]. Another recent concurrent and independent work [56] also focuses on BFV/BGV bootstrapping (using only BFV/BGV but not CKKS). Similar to all existing BFV/BGV works, this work focuses on removing the noise using BFV/BGV. With their elegant new techniques, their runtime is dependent on B where B is the number of digits of the noise that need to be removed, instead of $\log(p)$. Thus, their runtime is about 1-2x faster than [25] for the parameters tested in [25]. They additionally test some parameters that can hardly be tested using [25] and achieve 1-2 orders of magnitude improvement. While this improvement is impressive and they support regular BFV/BGV bootstrapping, their amortized runtime is still slower than ours. Furthermore, as discussed, we in addition support function evaluation during bootstrapping. Therefore, we believe these pros and cons can make our scheme favorable in some cases and theirs favorable in other cases. We give a concrete comparison in Section 6.

Comparison with [43]. One recent concurrent and independent work [43] also introduces a way to do functional bootstrapping for BGV/BFV. Similar to our functionality, they support a function evaluation while performing a bootstrapping. They support functions $\mathbb{Z}_p \to \mathbb{Z}_q$ where $q \neq p$, which means that their plaintext modulus before their functional bootstrapping is $\mathbb{Z}_p$ and plaintext modulus after is $\mathbb{Z}_q$. This is different from our functionality and may be suitable for different applications. Note that this also means that after one bootstrapping followed by a deep circuit, to perform a second bootstrapping, the application needs to perform either a regular bootstrapping or find some $q' \neq q$ and then use their bootstrapping.[5] Additionally, their runtime scales linearly in q, and thus their practical q choice is relatively small (e.g., $\mathbb{Z}_{17}$ as they show in their benchmarks). Lastly, like past works in regular BFV bootstrapping [33, 14, 26, 25, 59], they also explore some algebraic structures and thus require special parameter choices for BFV. Therefore, they can support much fewer slots than ours, thereby worse amortized efficiency.

1.2.3 Bootstrapping of other schemes

CKKS bootstrapping. CKKS bootstrapping is another line of work [13, 34, 47, 35, 5, 46, 42]. Similar to regular BFV bootstrapping, CKKS bootstrapping also only supports (approximate) identity function. Unlike BFV, CKKS instead computes with (approximate) real numbers. Therefore, their decryption process takes a different strategy from our construction or the BFV bootstrapping: they use sine to approximate a mod function, and then use a polynomial function to approximate the sine function.

[5] Via private communication with the authors.

	Supported functions	Depth	# of scalar multiplications	# of non-scalar multiplications	Plaintext space	# slots	Closeness (Definition 3.2)
Regular BFV Bootstrapping [33, 14, 26, 25, 59]	Identity function over the entire plaintext space	$\log(h) + \log\log(p^e)$	$\log_p(h) \cdot (\log_p(h) + e) \cdot p$	$(\log_p(h) \cdot (\sqrt{e}+\log_p(h)) \cdot \log p)$	$\mathsf{R}(p^e, d)$	N/d	N/A
			$\frac{(\log_p(h)\cdot(\log_p(h)+e)\cdot p)}{d}$	$(\log_p(h) \cdot (\sqrt{e}+\log_p(h)) \cdot \log p)/d$	$\mathbb{Z}_{p^e}$		
Our result	(1) $f_{\mathsf{pts}} : X \to Y$ $X, Y \subset \mathbb{Z}_t$ *Algorithm* 4	$\log(\lvert X\rvert \cdot r)$	$\lvert X\rvert \cdot r$	$\sqrt{\lvert X\rvert \cdot r}$	$\mathbb{Z}_t$	N	$\ell = \lvert\mathcal{Y}\rvert$, if $\lvert X\rvert = \frac{t-1}{r}$; no, o.w.
	(2) $f_{\mathsf{ranges}}(m) = y_i$ if $m \in [u_i, v_i]$ $u_i, v_i, y_i \in \mathbb{Z}_t, i \in [k], k \geq 2$ *Algorithm* 5	$\log(\sum_{i\in[k]}(\lvert v_i - u_i\rvert + r))$	$\sum_{i\in[k]}(\lvert v_i - u_i\rvert + r)$	$\sqrt{\sum_{i\in[k]}(\lvert v_i - u_i\rvert + r)}$			$\ell = 2$ with overhead (Remark 5.4)
	(3) $f_{\mathsf{ub}}(m) = y_i$ if $m \in [u_i, v_i]$ $u_i, v_i, y_i \in \mathbb{Z}_t, i \in [2]$ *Algorithm* 6	$\log(\lvert v_1 - u_1\rvert + r) + \log(t)$	1	$\lvert v_1 - u_1\rvert + r + \log(t)$			$\ell = 2$
Our result (Stepping stone)	(4) f_1 : identity function over $[0, t-1, r]$ *Algorithm* 1	$\log(t)$	t	$\sqrt{t}$			$\ell = 2$
	(5) $f_2 : [u, v, r'] \to Y$ $u, v, r' \in \mathbb{Z}_t, Y \subset \mathbb{Z}_t$ *Algorithm* 2	$\log(r(v-u)/r')$	$r(v-u)/r'$	$\sqrt{r(v-u)/r'}$			$\ell = 2$, if $r' = r$; $\ell = \lvert\mathcal{Y}\rvert$, o.w.

Table 1: Asymptotic behavior of our construction compared to prior works on regular BFV bootstrapping (ignoring some constants). $[a, b, c] := \{a, a + c, a + 2c, \ldots, b\}$ (i.e., the set of all the integers $x \in [a, b]$ such that $x - a$ divides c), where c divides $b - a$, p is some small prime satisfying $\mathsf{gcd}(p, m) = 1$, $e \geq 1$, the plaintext space is given as p^e, and d the multiplicative order of p in $\mathbb{Z}^*_{2N}$. t is some prime satisfying $t \equiv 1 \mod 2N$, and $r = O(\sqrt{h})$ is the modulus switching error range. Concretely speaking, for most practical parameters benchmarked in our work and prior works, $t \approx p^e$ (or $t \gg p^e$ for some parameters [59, 25]). h is the hamming weight of the secret key, and $r = O(\sqrt{h})$. At a high level, ℓ-closeness means that the output of all the out-of-the-range invalid inputs are mapped to the evaluation result of one of their closest ℓ valid inputs. Depth here means the multiplicative depth of the bootstrapping circuit.

(Batched) FHEW/TFHE bootstrapping. FHEW/TFHE bootstrapping [21, 18, 48] focuses on bootstrapping for a single LWE ciphertext. Recently, some works bootstrap multiple LWE ciphertexts at the same time, denoted as batched bootstrapping [57, 30, 58, 52, 53, 54]. This line of work also supports arbitrary function evaluation during bootstrapping. Looking ahead, some of our techniques can be applied to improve the batched bootstrapping method proposed in [54], as discussed in Section 8.

Functionality-wise, our major advantage over batched FHEW/TFHE bootstrapping is that our bootstrapping is embedded inside BFV circuits. One can easily perform multiplications and additions before or after our bootstrapping, which is inherently hard in the FHEW/TFHE case.

1.3 Paper organization

The rest of the paper is organized as follows. Section 2 introduces the notations and necessary background for the rest of the paper. Section 3 gives the formal definition of our generalized BFV functional bootstrapping. Section 4 describes the overall framework of our construction with two stepping-stone function families. Section 5 shows how our framework can be applied to more general function families. Section 6 includes the concrete efficiency of our construction and how it compares to prior works. Section 7 discusses some example applications that can take advantage of our generalized bootstrapping constructions. Section 8 presents how our technique can be used to improve batched FHEW/TFHE bootstrapping. Section 9 concludes the paper.

2 Preliminary

Let N be a power of two. Let $[u, v, r]$ denote the range from u to v with step value r (i.e., $[u, v, r] := \{u, u + r, u + 2r, \ldots, v\}$ and r divides $u - v$). Let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ denote the $2N$-th cyclotomic ring where N is a power-of-two, and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$ for some $Q \in \mathbb{Z}$. Let $[n]$ denote the set $\{1, \ldots, n\}$. Let $\vec{a}$ denote a vector and $\vec{a}[i]$ denote the i-th element of $\vec{a}$. Similarly, if A is a matrix, let $A[i][j]$ denote the element on the i-th row and j-th column of matrix A. Let $\|\vec{x}\|_\ell$ denote the ℓ-norm for vector $\vec{x}$ (calculated as $(\sum_{i\in|\vec{x}|}\vec{x}[i]^{\ell})^{1/\ell})$. If $x \in \mathcal{R}$, let $\|x\|_\ell$ denote the ℓ-norm of the coefficient vector of x, and let $x[i]$ denote the i-th coefficient of x.

Unless otherwise specified, the key is taken implicitly and correctly for functions (e.g., $\mathsf{Dec}(\mathsf{ct})$ where ct is some LWE ciphertext and Dec is the decryption procedure of LWE scheme).

All the divisions (i.e., a/b or $\frac{a}{b}$) and roundings (i.e., $\lceil\cdot\rfloor, \lceil\cdot\rceil, \lfloor\cdot\rfloor$) are performed in real numbers. All the other operations (including a^{-1}) are performed in finite field $\mathbb{Z}_t$ for some prime t (where t is specified if not obvious), unless otherwise noted.

2.1 B/FV Leveled Homomorphic Encryption

The BFV leveled homomorphic encryption scheme is first introduced in [6] using standard LWE assumption, and later adapted to ring LWE assumption by [22].

Given a polynomial $\in \mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$, the BFV scheme encrypts it into a ciphertext consisting of two polynomials, where each polynomial is from a larger cyclotomic ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ for some $Q > t$. We refer t as the plaintext modulus, Q as the ciphertext modulus, and N as the ring dimension. t satisfies that $t \equiv 1 \mod 2N$, where N is a power of two. [6]

Plaintext encoding. In practice, instead of having a polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$ to encrypt, applications usually have a vector of messages $\vec{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$. Thus, to encrypt such input messages, BFV first encodes it by constructing another polynomial $y(X) = \sum_{i\in[N]} y_i X^{i-1}$ where $m_i = y(\zeta_j)$, $\zeta_j := \zeta^{3^j} \mod t$, and ζ is the $2N$-th primitive root of unity of t. Such encoding can be done using an Inverse Number Theoretic Transformation (INTT), which is a linear transformation represented as matrix multiplication.

Encryption and decryption. The BFV ciphertext encrypting $\vec{m}$ under $\mathsf{sk} \leftarrow \mathcal{D}$ has the following format: $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, which satisfies $b - a \cdot \mathsf{sk} = \lfloor Q/t \rfloor \cdot y + e$ where $\lfloor Q/t \rfloor \cdot y \in \mathcal{R}_Q$ and y is the polynomial encoded in the way above, and e is a small error term sampled from a Gaussian distribution over $\mathcal{R}_Q$ with some constant standard deviation.

Symmetric key encryption can be done by simply sampling a random a and constructing b accordingly using sk. Public key encryption can also be achieved easily but it is not relevant to our paper so we refer the readers to prior works (e.g., [6, 22, 40]) for details.

Decryption is thus calculating $y' \leftarrow \lceil (t/Q) \cdot (b - a \cdot \mathsf{sk}) \rfloor \in \mathcal{R}_t$ (note that $(b - a \cdot \mathsf{sk})$ is done over $\mathcal{R}_Q$), and then decodes it by applying a procedure to revert the encoding process (which is also a linear transformation). For simplicity, we assume $\mathsf{BFV.Dec}$ also embeds the decoding procedure and thus outputs plaintext $y' \in \mathbb{Z}_t^N$ in the decoded form directly (instead of a polynomial $y \in \mathcal{R}_t$). Similarly, we assume $\mathsf{BFV.Enc}$ contains the encoding process, thus taking a plaintext $y' \in \mathbb{Z}_t^N$. In addition, define $\mathsf{PartialDec}(\mathsf{sk}, \mathsf{ct} = (a, b) \in \mathcal{R}_Q^2) := b - a \cdot \mathsf{sk} \in \mathcal{R}_Q$ (i.e., decryption without performing the rounding to $\mathcal{R}_t$).

BFV operations. BFV essentially supports addition, multiplication, rotation, and polynomial function evaluation, satisfying the following property:

- (Addition) $\mathsf{BFV.Dec}(\mathsf{ct}_1 + \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) + \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Multiplication) $\mathsf{BFV.Dec}(\mathsf{ct}_1 \times \mathsf{ct}_2) = \mathsf{BFV.Dec}(\mathsf{ct}_1) \times \mathsf{BFV.Dec}(\mathsf{ct}_2)$
- (Rotation) $\mathsf{BFV.Dec}(\mathsf{rot}(\mathsf{ct}, j))[i] = \mathsf{BFV.Dec}(\mathsf{ct})[i + j \pmod N)], \forall i, j \in [N]$
- (Polynomial evaluation) $\mathsf{BFV.Dec}(\mathsf{BFV.Eval}(\mathsf{ct}, f)) = f(\mathsf{BFV.Dec}(\mathsf{ct}))$, where $f : \mathbb{Z}_t \to \mathbb{Z}_t$ is a polynomial function. Note that this is implied by addition and multiplication.
- (Vector-matrix multiplication) $\mathsf{BFV.Dec}(\mathsf{ct} \times A) = \mathsf{BFV.Dec}(\mathsf{ct}) \times A$, where $A \in \mathbb{Z}_t^{N\times D}$ for any $D > 0$.

[6]Note that this is the relationship between t, N does not need to be satisfied in general (e.g., see [31, 32] for the general encoding). However, throughout our paper, we suppose it holds to maximize the concrete efficiency and thus introduce it this way for simplicity.

Given a BFV ciphertext ct and its corresponding secret key sk, we also assume that its noise budget can be derived via interface $\mathcal{B}(\mathsf{sk}, \mathsf{ct})$. A noise budget is essentially the gap between the plaintext encrypted under ct and the noise inside ct, which is used to allow operations (e.g., multiplications) over the ciphertexts. For simplicity, $\mathcal{B}$ with subscripts is also used to refer to hardcoded noise budget bounds (e.g., $\mathcal{B}_{\mathsf{in}}$ represents the noise budget requirement of the input).

All operations are operated over the entire plaintext vector $m \in \mathbb{Z}_t^N$ (element-wise). Thus, all messages need to be evaluated using the same polynomial f by default. This is also known as the Single Instruction Multiple Data (SIMD) property of BFV. Note that vector-matrix multiplication can be realized using scalar multiplication (implied by addition) and rotation. All of these BFV operations are used as blackboxes in our main constructions and we refer the readers to [6, 22, 40, 33, 31] to see how these operations are accomplished in detail. In this paper, we sometimes directly refer to the interfaces (e.g., Dec) for short without the BFV prefix (e.g., $\mathsf{BFV.Dec}$).

3 Definition of Generalized BFV Bootstrapping

We first define a more general BFV bootstrapping procedure.[7] As discussed in Section 1, the main goal of this generalized definition is to capture (1) the relaxation that not the entire plaintext space needs to be valid, and for the invalid plaintexts, the corresponding correctness does not need to be guaranteed by the construction; (2) the bootstrapping itself contains an evaluation of a given function, thus making the bootstrapping procedure itself more useful and can be embedded directly into the circuit without inducing stand-alone bootstrapping overhead. These two properties are captured as follows: given a function $f : \mathcal{X} \to \mathcal{Y}$ and input ciphertext encrypting $x \in \mathcal{X}$, after the bootstrapping procedure, the output ciphertext encrypts $y = f(x) \in \mathcal{Y}$. If $x \notin \mathcal{X}$, the output is not defined, and thus can be arbitrarily decided by the construction.

The definition also captures the most basic requirement of bootstrapping: the output ciphertext has more noise budget (or equivalently less noise) compared to the input ciphertext, such that bootstrapping can be used to support circuits with arbitrary depth.

Formally, the general BFV bootstrapping procedure is defined as follows, consisting of two PPT algorithms:

- $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}, \mathsf{pp}_{\mathsf{aux}}), \mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{Setup}(1^\lambda)$: Setup takes a security parameter λ, and outputs a secret key sk, a bootstrapping key btk, and a public parameter pp including ring dimension N, plaintext space t, input noise budget $\mathcal{B}_{\mathsf{in}}$, output noise budget $\mathcal{B}_{\mathsf{out}}$, a function family $\mathcal{F}$, and auxiliary public parameters $\mathsf{pp}_{\mathsf{aux}}$.
- $\mathsf{ct}' \leftarrow \mathsf{Boot}(\mathsf{pp}, \mathsf{btk}, f, \mathsf{ct})$: takes the public parameter pp, a bootstrapping key btk, a function $f \in \mathcal{F}$, a ciphertext ct and outputs a ciphertext ct'.

Definition 3.1 (Correctness)**.** The bootstrapping procedure is correct, if it satisfies the following: let $(\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}, \mathsf{pp}_{\mathsf{aux}}), \mathsf{sk}, \mathsf{btk}) \leftarrow \mathsf{Setup}(1^\lambda)$, for any function $f : \mathcal{X} \to \mathcal{Y} \in \mathcal{F}$ (where $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{Z}_t$ and $|\mathcal{X}| \geq |\mathcal{Y}| \geq 2$),[8] any honest input ciphertext ct with $\mathcal{B}(\mathsf{sk}, \mathsf{ct}) \geq \mathcal{B}_{\mathsf{in}}$, let $\mathsf{ct}' \leftarrow \mathsf{Boot}(\mathsf{pp}, \mathsf{btk}, f, \mathsf{ct})$, $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \in \mathbb{Z}_t^N$, $\vec{m}' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') \in \mathbb{Z}_t^N$, it holds that:

$$\Pr\left[\begin{array}{ll} & \forall\, i \in [N],\ \text{if } \vec{m}[i] \in \mathcal{X}, f(\vec{m}[i]) = \vec{m}'[i] \\ \wedge & \mathcal{B}(\mathsf{sk}, \mathsf{ct}') \geq \mathcal{B}_{\mathsf{out}} > \mathcal{B}_{\mathsf{in}} \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

In some cases, applications may require that even if $x \notin \mathcal{X}$, the result does not "deviate" too much (such that the error can be predicted and algorithmically handled). To capture this demand, we define an additional property we call "ℓ-closeness". Essentially, it means that even if $x \notin \mathcal{X}$, the output ciphertext encrypts $y \in \mathcal{S}$, where $\mathcal{S} \subseteq \mathcal{Y}$ contains the evaluation results of the ℓ points of $\mathcal{X}$ that are the "closest" to x (for a point $x' \in \mathcal{X}$, the smaller $|x - x'|$ is, the closer x' and x are).

[7]We focus on BFV in this work, but all our results can be directly transformed to BGV with minimum modification (e.g., with techniques in [3, Sec A]).

[8]For $|\mathcal{Y}| = |[y]| = 1$, a trivial yet valid bootstrapping is to directly output a BFV ciphertext with all slots encrypting y.

Note that this property is auxiliary to the regular correctness, and may not be needed in some applications (see Section 7 for discussion). Looking ahead, some of our constructions achieve the closeness property while some do not. The ones that do not achieve it take advantage of such further relaxation to achieve even better efficiency. With these in mind, we define ℓ-closeness formally as follows.

Definition 3.2 (ℓ-closeness)**.** The bootstrapping procedure is ℓ-close, if it satisfies the following: for the same quantifiers as correctness; for all $x \in \mathbb{Z}_t \setminus \mathcal{X}$, let $y_{x,1}, \ldots, y_{x,|\mathcal{Y}|}$ denote all the points in $\mathcal{Y}$ satisfying $|f_x^{-1}(y_{x,1}) - x| \leq |f_x^{-1}(y_{x,2}) - x| \leq \cdots \leq |f_x^{-1}(y_{x,|\mathcal{Y}|}) - x|$,[9][10] and $\mathcal{S}_x := \{y_{x,1}, \ldots, y_{x,\ell}\}$; it holds that for all $i \in [N]$, if $\vec{m}[i] \notin \mathcal{X}$:

$$\Pr\left[f(\vec{m}[i]) \in \mathcal{S}_{\vec{m}[i]}\right] > 1 - \mathsf{negl}(\lambda)^{11}$$

Remark 3.1. The regular BFV bootstrapping, which only supports $\mathcal{F} = \{I\}$ with $I : \mathbb{Z}_t \to \mathbb{Z}_t$ being the identity function, is a special case of our definition.

Remark 3.2. Naturally, we want at least $\mathcal{B}_{\mathsf{out}} > \mathcal{B}_{\mathsf{in}} + \mathcal{B}_{\times}$, where $\mathcal{B}_{\times}$ is the noise budget needed for one multiplication. Thus, after every bootstrapping, the output ciphertext can perform at least one multiplication (which implies one addition for BFV) before the next bootstrapping. This gives us the ability to evaluate a circuit with arbitrary depth. However, for generality, we simply require $\mathcal{B}_{\mathsf{out}} > \mathcal{B}_{\mathsf{in}}$, the minimum requirement for a non-trivial bootstrapping scheme, and leave the value of $\mathcal{B}_{\mathsf{out}} - \mathcal{B}_{\mathsf{in}}$ to be tuned based on applications during setup.

Also, interestingly, even $\mathcal{B}_{\mathsf{out}} \leq \mathcal{B}_{\mathsf{in}}$, as long as $\mathcal{F}$ is not purely the identity function, the construction may not be trivial. For example, if $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}}$ and $\mathcal{F} = \{f(x) = x^2\}$, essentially this construction allows free squaring. However, we do not capture this case in our definition as it introduces extra complexity. However, this may be of its own interest and worth exploring.

Remark 3.3. Again, we believe that ℓ-closeness may not be necessary for some applications. A construction that only satisfies correctness while outputting an arbitrary value for invalid inputs can also be interesting and applicable in many senarios (Section 7).

Furthermore, even if a construction does not satisfy ℓ-closeness, the output for the invalid inputs can still be "structured". For example, if input is not in $\mathcal{X}$, the scheme always outputs 0, while $0 \notin \mathcal{Y}$. This is still "predictable" but not captured by the closeness property. How to define such a more general property is left for future works with constructions having such properties.

4 Our General Framework for Bootstrapping

In this section, we propose a (relaxed) BFV bootstrapping framework. In Section 4.1, we start with a simple function (the identity function over a subset of the plaintext space) to show how the general framework works. Then in Section 4.2, we use a generalized type of function to show how the framework can be used for more versatile functions. These two types of functions work as stepping stones to fully introduce our framework. We later show in Section 5 how the framework works for more general types of function families.

4.1 Bootstrap for Identity Function f_1 over $[0, t-1-r, r]$

As a stepping stone, let us first consider the identity function. Different from prior works that focus on identity mapping on all values in $\mathbb{Z}_t$, we define f_1 with input consisting of a set of points $\mathcal{X} \subset \mathbb{Z}_t$. This allows us to construct a more efficient bootstrapping scheme.

Let $\mathcal{X} := [0, t-1-r, r]$, for some $1 \leq r < t$ (r to-be-fixed later). Denote $\vec{m} \in \mathbb{Z}_t^N \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{in}})$, where $\mathsf{ct}_{\mathsf{in}}$ is the input ciphertext, sk is the corresponding secret key, t is the plaintext space, and N is the

[9]Let $f_x^{-1}(y)$ denote a point z where $f(z) = y$ and $z - x = \min_{z' \in \mathcal{X}, f(z') = y}(z' - x)$. In other words, $f_x^{-1}(y)$ outputs a point that is (1) a valid input in $\mathcal{X}$; and (2) is the close to x among all possible points z' satisfying $f(z') = y$.

[10]If $|f_x^{-1}(y_{x,i}) - x| = |f_x^{-1}(y_{x,j}) - x|$ for $i \neq j$, then any order is accepted.

[11]The randomness is taken over the input ciphertext and the generated keys.

ring dimension. Our goal is to compute $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{Boot}(\cdot, \cdot, \mathsf{ct}_{\mathsf{in}}, f_1)$ such that for all $i \in [N]$, if $\vec{m}[i] \in \mathcal{X}$, $\vec{m}'[i] = \vec{m}[i]$, where $\vec{m}' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}})$.

Decoding the input ciphertext. Recall that as introduced in Section 2, to encrypt a message $\vec{m} \in \mathbb{Z}_t^N$, BFV constructs a ciphertext ct that encrypts a polynomial $y(X)$ encoding $\vec{m}$. Formally speaking, let $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, it holds that $b - a \cdot \mathsf{sk} \approx \lfloor Q/t \rfloor \cdot y$, where $y(X) = \sum_{i \in [N]} y[i] X^{i-1} \in \mathcal{R}_t$, satisfying $\vec{m}[i] = y(\zeta_i)$ (where $\zeta_i := \zeta^{3^i}$) for all $i \in [N]$ (ζ is the $2N$-th primitive root of unity of t). Thus, the very first step for bootstrapping (i.e., homomorphic decryption) is to perform a decoding, homomorphically changing the encrypted $y(X)$ into $m(X) := \sum_{i \in [N]} \vec{m}[i] X^{i-1}$ by computing $\mathsf{ct}_1 \leftarrow \mathsf{ct} \cdot U^\intercal$ homomorphically with:

$$U := \begin{pmatrix} 1 & \zeta_0 & \zeta_0^2 & \cdots & \zeta_0^{N-1} \\ 1 & \zeta_1 & \zeta_1^2 & \cdots & \zeta_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta_{\frac{N}{2}-1} & \zeta_{\frac{N}{2}-1}^2 & \cdots & \zeta_{\frac{N}{2}-1}^{N-1} \\ 1 & \bar{\zeta}_0 & \bar{\zeta}_0^2 & \cdots & \bar{\zeta}_0^{N-1} \\ 1 & \bar{\zeta}_1 & \bar{\zeta}_1^2 & \cdots & \bar{\zeta}_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\zeta}_{\frac{N}{2}-1} & \bar{\zeta}_{\frac{N}{2}-1}^2 & \cdots & \bar{\zeta}_{\frac{N}{2}-1}^{N-1} \end{pmatrix} \in \mathbb{Z}_t^{N \times N}$$

as the $\mathsf{SlotToCoeff}$ step in [54].

Switching modulus. Now we have a ciphertext $\mathsf{ct}_1 = (a_1, b_1) \in \mathcal{R}_Q^2$, encrypting $m(X) \in \mathcal{R}_t$ defined above. Recall that the plaintext space of the underlying BFV scheme is $\mathbb{Z}_t$. Therefore, to homomorphically decrypt ct_1, we need to first match the modulus by performing a modulus switching: $\mathsf{ct}_2 \leftarrow \lceil t \cdot (a_1, b_1)/Q \rfloor \in \mathcal{R}_t^2$. Notice that with $\mathsf{ct}_1 = (a_1, b_1)$ satisfying $b_1 - a_1 \cdot \mathsf{sk} = \alpha \cdot m + e$ (where $\alpha = \lfloor Q/t \rfloor$), for some small noise term $e \in \mathcal{R}_Q$. After modulus switching, we have $\mathsf{ct}_2 = (a_2, b_2)$ satisfying $b_2 - a_2 \cdot \mathsf{sk} = m + e'$, where $e'(X) := \sum_{i \in [N]} e'[i] X^{i-1} \in \mathcal{R}_t$ is some noise term dominated by the error introduced through modulus switching, which might "contaminate" the correct message m.

Fortunately, we do not need to correctly decrypt all possible values in $\mathbb{Z}_t$, but instead, only consider the correct decryption of $m[i] \in [0, t-1-r, r]$; for invalid values in $\mathbb{Z}_t \setminus [0, t-1-r, r]$, we do not need to guarantee the correctness per Definition 3.1. Therefore, we fix r to be the smallest positive integer such that $\Pr[\|e'[i]\| < r/2] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in [N]$.[12]

This means that for $m[i] \in [0, t-1-r, r]$, $m[i] + e'[i] \in (m[i] - r/2, m[i] + r/2)$, Rounding $m[i] + e'[i]$ to the nearest value in $[0, t-1-r, r]$ then gives us exactly $m[i]$, which provides the correct decryption. Formally speaking, with $\mathsf{ct}_2 = (a_2, b_2)$, let $m'(X) := \sum m'[i] X^{i-1} \leftarrow b_2 - a_2 \cdot \mathsf{sk}$, it holds that $r \left\lceil \frac{m'[i]}{r} \right\rfloor = \vec{m}[i] \in \mathbb{Z}_t$, for all $i \in [N]$, except with negligible probability. With these, we proceed to introduce how the homomorphic decryption is done.

Analysis of t and r. One may wonder whether this t is always possible to achieve given that we need $t > r$. Luckily, this is easy: since the modulus switching error, as mentioned in [21, Lemma 5], is $O(\sqrt{h})$ where h is the hamming weight of the secret key.[13] Thus, we simply need to set $t = \omega(\sqrt{h})$. To utilize the full SIMD power of the BFV scheme, one needs to set $t > N$ such that $t \equiv 1 \mod 2N$ (as discussed in Section 2), and thus $t = \Omega(N) = \omega\sqrt{h}$ (for ternary or binary secret keys). Note that prior works [33, 14, 26, 25, 59] similarly require the keys to be ternary or binary (or more commonly a sparse key with some fixed hamming weight), as they need to bound the wrap-around over $\mathbb{Z}_t$ as well to perform the digit extraction method. Furthermore, most existing implementations of BFV [61, 4, 1] use a ternary secret key. Thus, we believe our parameter setting is easily achievable.

Homomorphic decryption. The final step is to homomorphically decrypt ct_2. Note that now $\mathsf{ct}_2 \in \mathcal{R}_t^2$, and the plaintext modulus is t. Therefore, we can simply homomorphically compute $b_2 - a_2 \cdot \mathsf{sk}$ over $\mathbb{Z}_t$ by

[12] For simplicity, we assume r divides $t-1$. This is w.l.o.g because we can make the range $[0, t-t'-r, r]$ where $t-t'$ is the largest multiple of r with $t' > 0$. This change does not affect the main point or technique of this paper.

[13] Note that this is the modulus switching error of the LWE ciphertexts, which we can achieve by simply transforming one RLWE ciphertext to N LWE ciphertexts using the $\mathsf{SampleExtract}$ procedure as discussed incite [18]. One may also simply bound the modulus switching error of RLWE as discussed in [40], which is $O(\sqrt{N})$ for binary/ternary secrets.

utilizing the free mod operation. Compared to prior works [33, 14, 26], which need to perform plaintext space switching, our construction is much simpler. In more detail, our homomorphic decryption is carried out in two steps:

- First, given $\mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}_t^2$, we evaluate a partial decryption process $\mathsf{PartialDec}(\mathsf{sk}, \mathsf{ct}_2)$, which computes $b_2 - \mathsf{ct}_{\mathsf{sk}} \times A_2$, where $\mathsf{ct}_{\mathsf{sk}}$ is the encrypted sk under BFV, $A_2 := \begin{pmatrix} a_2[1] & a_2[2] & a_2[3] & \dots & a_2[N] \\ -a_2[N] & a_2[1] & a_2[2] & \dots & a_2[N-1] \\ -a_2[N-1] & -a_2[N] & a_2[1] & \dots & a_2[N-2] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_2[2] & -a_2[3] & -a_2[4] & \dots & a_2[1] \end{pmatrix} \in \mathbb{Z}_t^{N \times N}$, and $\mathsf{ct}_{\mathsf{sk}} \times A_2$ is homomorphically computed as a vector-matrix multiplication. The resulting ciphertext is denoted as ct_3.
- With ciphertext ct_3 encrypting $(m'[1], \dots, m'[N]) \in \mathbb{Z}_t^N$ (recall that $m' = m + e'$ for some small error e'), we then simply need to compute $r \left\lceil \frac{m'_i}{r} \right\rfloor$ over $\mathbb{Z}_t$ for all $i \in [N]$. This can be done by interpolating a function $f_{\mathsf{post}}(x) : \mathbb{Z}_t \to \mathbb{Z}_t$, s.t., for all $x \in \mathbb{Z}_t$, $f_{\mathsf{post}}(x) = r \left\lceil \frac{x}{r} \right\rfloor$ via Lagrange interpolation. The resulting ciphertext, denoted as $\mathsf{ct}_{\mathsf{out}}$, encrypts the same message as $\mathsf{ct}_{\mathsf{in}}$ as desired.

Bootstrapping key and noise setup. Lastly, we discuss what the bootstrapping key contains. Since we need to homomorphically decrypt the ciphertext ct_2, we need to include $\mathsf{ct}_{\mathsf{sk}}$ which is the encrypted sk under BFV. Moreover, BFV public keys pk and BFV evaluation keys evk, the keys needed to evaluate the circuits in the construction (e.g., the relinearization key and the rotation keys[14]), are all included in the bootstrapping key.

We also need to specify the input noise budget and output noise budget. $\mathcal{B}_{\mathsf{in}}$ is set to be enough for evaluating the $\mathsf{SlotToCoeff}$ step, and Q to be large enough to evaluate f_{post} such that afterwards there are still at least $\mathcal{B}_{\mathsf{out}} > \mathcal{B}_{\mathsf{in}}$ noise budget left.

To finalize the algorithm of BFV bootstrapping for our identity function f_1, we need to do some preparation work in the Setup phase, including choosing all public parameters such as the ring dimension N and the plaintext space t. The bootstrapping keys are generated as discussed above. Finally, we define $f_{\mathsf{post}}(x) := r \left\lceil \frac{x}{r} \right\rfloor$. The procedure is formalized in Algorithm 1.

Theorem 4.1. Algorithm 1 is a correct BFV functional bootstrapping (Definition 3.1) procedure with function family $\mathcal{F} := \{f(x) = x, \forall x \in [0, t-1-r, r]\}$ where t, r are from pp generated by Setup, assuming the correctness of BFV. Furthermore, it is 2-close (Definition 3.2).

Proof. Given that the underlying BFV is correct (i.e., all the homomorphic evaluations are completed as expected given enough noise budget), let $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{in}}) \in \mathbb{Z}_t^N$, $\mathsf{ct}_1 = (a_1, b_1) \in \mathcal{R}_Q^2$, $m_1 \leftarrow \lceil (t/Q)(b_1 - a_1 \cdot \mathsf{sk}) \rfloor$ (i.e., BFV decryption without the decoding process), it holds that $m_1 = \sum_{i \in [N]} \vec{m}[i] X^{i-1} \in \mathcal{R}_t$, by condition (2) (that there is enough noise budget for $\mathsf{SlotToCoeff}$). Let $\mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}_t^2$, $m_2 := \sum_{i \in [N]} m_2[i] X^{i-1} \leftarrow b_2 - a_2 \cdot \mathsf{sk} \in \mathcal{R}_t$, Then, it holds that $\Pr[m_2[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2)] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in [N]$, by condition (3) (that the error range r is large enough). Thus, let $\vec{m}_3 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_3) \in \mathbb{Z}_t^N$, for all $i \in [N]$, $\vec{m}_3[i] = m_2[i]$. Lastly, let $\vec{m}_4 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) \in \mathbb{Z}_t^N$, we have $\vec{m}_4[i] = r \cdot \left\lceil \frac{\vec{m}_3[i]}{r} \right\rfloor = r \cdot \left\lceil \frac{m_2[i]}{r} \right\rfloor$ for all $i \in [N]$ by f_{post} and condition (3) (that there is enough noise budget to evaluate $\mathsf{PartialDec}$ and f_{post}). Since we have $m_2[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2)$, then if $\vec{m}[i] \in [0, t-1-r, r]$, we have $\vec{m}_4[i] = \vec{m}[i]$ for all $i \in [N]$.

The 2-closeness property is straightforward. The intuition is that the invalid input points are "rounded" to the two nearest valid input points (with some biased probability per the $\mathsf{ModSwitch}$ error introduced in line 16). In more detail, let $x = \vec{m}[i]$, and let $z_{x,i} \leftarrow f_x^{-1}(y_{x,i})$ for $i \in [1, 2]$ and $y_{x,i}$ in Definition 3.2; let $d_1 \leftarrow x - z_{x,1}$, $d_2 \leftarrow x - z_{x,2}$, (where $z_{x,j}$ are per closeness definition). Note that we have $d_2 = r - d_1$ and $d_2 \geq r/2 \geq d_1$. For 2-closeness to not hold, $\mathsf{err}(\mathsf{sk}, \mathsf{ct}_3) > r/2 + d_1$, which happens with $\mathsf{negl}(\lambda)$ by condition (3). □

[14]For simplicity, we assume all possible rotation keys are generated. Later, we discuss how to only generate the necessary ones.

Algorithm 1 BFV Bootstrapping for $f_1 : [0, t-1-r, r] \to [0, t-1-r, r]$

```
1: procedure Setup($1^\lambda$)
2:     Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of Boot below:
3:     (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
4:     (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate SlotToCoeff.
5:     (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating PartialDec followed by $f_{\mathsf{post}}$, still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.    ▷ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
6:     Let $r$ be the error bound such that $\Pr[|\mathsf{err}(\mathsf{sk}, \mathsf{ct}_3)| < r/2] \geq 1 - \mathsf{negl}(\lambda)$, where $\mathsf{ct}_3$ is in line 16 below.
7:     Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
8:     $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}})$
9:     $\mathcal{F}_1 := \{f_1(x) := x \text{ iff } x \in [0, t-1-r, r]\}$
10:    return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_1, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
11: procedure Boot($\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_1$)
12:    If $f_1 \notin \mathcal{F}_1$, abort.
13:    $f_{\mathsf{post}}(x) := r \left\lceil \frac{x}{r} \right\rfloor$
14:    $\mathsf{ct}_1 \leftarrow \mathsf{ct}_{\mathsf{in}} \times U^{\mathsf{T}}$ (evaluated homomorphically)
15:                    ▷ Recall that $U$ is defined in Section 4.1 and this step is SlotToCoeff
16:    $\mathsf{ct}_2 \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_1, t)$
17:    Parse $\mathsf{ct}_2 = (a_2 = \sum_{i \in [N]} a_{2,i} X^{i-1}, b_2 = \sum_{i \in [N]} b_{2,i} X^{i-1}) \in \mathcal{R}_t^2$
18:    Let $A_2 := \begin{pmatrix} a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,N} \\ -a_{2,N} & a_{2,1} & a_{2,2} & \cdots & a_{2,N-1} \\ -a_{2,N-1} & -a_{2,N} & a_{2,1} & \cdots & a_{2,N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_{2,2} & -a_{2,3} & -a_{2,4} & \cdots & a_{2,1} \end{pmatrix} \in \mathbb{Z}_t^{N \times N}$, and $\vec{b}_2 \leftarrow (b_{2,i})_{i \in [N]} \in \mathbb{Z}_t^N$
19:    $\mathsf{ct}_3 \leftarrow \vec{b}_2 - \mathsf{ct}_{\mathsf{sk}} \times A_2$ (evaluated homomorphically)    ▷ i.e., PartialDec($\mathsf{ct}_2, \mathsf{sk}$)
20:    $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{BFV.Eval}(\mathsf{evk}, \mathsf{ct}_3, f_{\mathsf{post}})$
21:    return $\mathsf{ct}_{\mathsf{out}}$.
```

Biased rounding for invalid inputs. In addition to 2-closeness, there is another property of our construction with respect to invalid inputs. At a high level, an invalid input rounds to the nearest valid input with high probability p and rounds to the second nearest valid input with $1-p$, where $p \gg 1-p$ as long as the invalid input is obviously closer to one input than the other (i.e., $d_1 \ll d_2$ using the notations in the proof). In more details, recall that $\Pr[\vec{m}_4[i] = f(z_{x,1})] \leq \Pr[\vec{m}_4[i] = f(z_{x,2})]$ for all $x = \vec{m}[i] \notin \mathcal{X}$. W.l.o.g, assume $z_{x,1} = z_{x,2} + r$. Then, if the error introduced in line 16 is $\geq d_1 - r/2 \in \mathbb{Z}$, $\vec{m}_2[i] \in [z_{x,1} - r/2, z_{x,1} + r/2)$; otherwise, $\vec{m}_2[i] \in [z_{x,2} - r/2, z_{x,2} + r/2)$. Therefore, $\Pr[\vec{m}_4[i] = f(z_{x,1})] < \Pr[\vec{m}_4[i] = f(z_{x,2})]$ (the same argument works when $z_{x,1} = z_{x,2} - r$). Note that since the error is Gaussian, if $d_1 << d_2$, $\Pr[\vec{m}_4[i] = f(z_{x,1})] \ll Pr[\vec{m}_4[i] = f(z_{x,2})]$. If $d_1 \approx d_2$, the probability becomes close.

4.2 Bootstrapping for $f_2 : [u, v, r'] \to \mathcal{Y}$

We now extend the above identity function into a more general function family: $\mathcal{F}_2 = \{f_2 : [u, v, r'] \to \mathcal{Y}\}$, where $u, v, r' \in \mathbb{Z}_t$, and $\mathcal{Y}$ being any subset of $\mathbb{Z}_t$ with $|\mathcal{Y}| \leq |[u, v, r']| = \frac{t-1}{r}$ ($[0, t-1-r, r]$ has $\frac{t-1}{r}$ points). [15] It is easy to see that f_1 is the special form with $u = 0, v = t-1-r, r' = r$ and $\mathcal{Y} = [u, v, r']$.

Preprocess the input ciphertext with f_{pre}. The very first challenge is that if we have $r' < r$ (call that r is set to be the error bound of modulus switching), after multiplying the ciphertext with t/Q during the

[15]Note that if $|[u, v, r']| < \frac{t-1}{r}$, we can either pad dummy elements to follow the same bootstrapping procedure or apply a more efficient way, introduced in Section 5.1.2.

modulus switching step, the encrypted messages will be contaminated by the error incurred and thus the decryption process fails.

To resolve this issue, we first "preprocess" the input ciphertexts by stretching the small intervals r' to be r the error bound: in this case the encrypted messages would survive the modulus switching procedure. In more detail, we construct a bijective mapping $f_{\mathsf{pre}}(x) : [u, v, r'] \to [0, t-1-r, r]$, defined as $f_{\mathsf{pre}}(x) := (x-u) \cdot r \cdot (r')^{-1}$. Before we perform the original $\mathsf{SlotToCoeff}$ process as the first step discussed above, we first homomorphically evaluate f_{pre} over the input ciphertext $\mathsf{ct}_{\mathsf{in}}$ (which means by the SIMD property we evaluate $f_{\mathsf{pre}}(\vec{m}[i])$ for all $i \in [N]$ and $\vec{m}$ being the message vector encrypted under $\mathsf{ct}_{\mathsf{in}}$).

A new f_{post} function. As before, after preprocessing, we perform $\mathsf{SlotToCoeff}$ and modulus switching. The resulting ciphertext encrypts $\vec{m}' \in \mathbb{Z}_t^N$ such that $r \cdot \lceil \vec{m}'[i]/r \rfloor = f_{\mathsf{pre}}(\vec{m}[i])$ for all $i \in [N]$. Here comes the second challenge: instead of simple identity mapping, which requires nothing else other than output $\vec{m}'$ in our previous construction [16], $\mathcal{Y}$ as the output set of f_2 can be any arbitrary subset of $\mathbb{Z}_t$ with size $\leq |\frac{t-1}{r'}|$.

Thus, we need a new function $f_{\mathsf{post}}(x) : \mathbb{Z}_t \to \mathbb{Z}_t$ to map $\vec{m}'$ onto the corresponding values in $\mathcal{Y}$, i.e., $f_{\mathsf{post}}(x) = f_2(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$. Note that since f_{pre} is bijective, f_{pre}^{-1} always exists. The correctness is as follows:

$$f_{\mathsf{post}}(\vec{m}'[i]) = f_2(f_{\mathsf{pre}}^{-1}(r \cdot \lceil \vec{m}'[i]/r \rfloor)) = f_2(f_{\mathsf{pre}}^{-1}(f_{\mathsf{pre}}(\vec{m}[i]))) = f_2(\vec{m}[i])^{17}$$

In summary, most of the bootstrapping process for this new function family remains the same as for the identity function, and the only parts that differ are the new function f_{pre} and a revised f_{post} function.

Regarding the Setup phase, we set $\mathcal{B}_{\mathsf{in}}$ to be large enough to evaluate *any* degree one function (to accommodate the noise growth in the worst case), since f_{pre} is at most a degree-1 polynomial. Similarly, for f_{post} to be at most degree-$(t-1)$, we set Q large enough to accommodate an arbitrary degree-$(t-1)$ function. We formalize our construction in Algorithm 2.

Theorem 4.2. Algorithm 2 is a correct BFV functional bootstrapping (Definition 3.1) procedure for function family $\mathcal{F}_2 := \{f_2 := [u, v, r'] \to \mathcal{Y} \mid (u, v, r' \in \mathbb{Z}_t) \wedge (\mathcal{Y} \subset \mathbb{Z}_t) \wedge (|\mathcal{Y}| \leq |[u, v, r']| = \frac{t-1}{r})\}$, assuming the correctness of BFV. Furthermore, if it $|\mathcal{Y}|$-closeness (Definition 3.2).

Proof. We prove this the same way as in Theorem 4.1. Given that the underlying BFV is correct (i.e., all the homomorphic evaluations are completed as expected given enough noise budget), let $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{in}}) \in \mathbb{Z}_t^N$ and $\vec{m}_1 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) \in \mathbb{Z}_t^N$, we have $\vec{m}_1 = f_{\mathsf{pre}}(\vec{m})$. Then, let $\mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}_Q^2$, $m_2 \leftarrow \lceil (t/Q)(b - a_2 \cdot \mathsf{sk}) \rfloor$, we have $m_2 = \sum_{i \in [N]} \vec{m}_1[i] X^{i-1} \in \mathcal{R}_t$, by condition (2) (that there is enough noise budget for f_{pre} and $\mathsf{SlotToCoeff}$). Let $\mathsf{ct}_3 = (a_3, b_3) \in \mathcal{R}_t^2$, $m_3 := \sum_{i \in [N]} m_3[i] X^{i-1} \leftarrow b_3 - a_3 \cdot \mathsf{sk} \in \mathcal{R}_t$, Then, it holds that $\Pr[m_3[i] \in (\vec{m}_1[i] - r/2, \vec{m}_1[i] + r/2)] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in [N]$, by condition (3) (that the error range r is large enough). Thus, let $\vec{m}_4 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_4) \in \mathbb{Z}_t^N$, for all $i \in [N]$, $\vec{m}_4[i] = m_3[i]$. Lastly, let $\vec{m}_5 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) \in \mathbb{Z}_t^N$, we have $\vec{m}_5[i] = f_2(f_{\mathsf{pre}}^{-1}(r \cdot \left\lceil \frac{\vec{m}_4[i]}{r} \right\rfloor)) = f_2(f_{\mathsf{pre}}^{-1}(r \cdot \left\lceil \frac{m_3[i]}{r} \right\rfloor))$ for all $i \in [N]$ by f_{post} and condition (5) (that there is enough noise budget to evaluate $\mathsf{PartialDec}$ and f_{post}). Since $m_3[i] \in (\vec{m}_1[i] - r/2, \vec{m}_1[i] + r/2)$, we have $r \cdot \left\lceil \frac{m_3[i]}{r} \right\rfloor = \vec{m}_1[i]$ for all $i \in [N]$, and thus $f_{\mathsf{pre}}^{-1}(r \cdot \left\lceil \frac{m_3[i]}{r} \right\rfloor) = f_{\mathsf{pre}}^{-1}(\vec{m}_1[i]) = \vec{m}[i]$. Therefore, $\vec{m}_5[i] = f_2(\vec{m}[i])$ as required.

$|\mathcal{Y}|$-closeness is straightforward: not that for any $\vec{m}[i] \notin \mathcal{X}$, $\vec{m}_4[i] \in \mathbb{Z}_t$. Furthermore, f_{post} takes all possible values of $\mathbb{Z}_t$ and maps it to $\mathcal{Y}$. Therefore, $\vec{m}_5[i] \in \mathcal{Y}$ and $|\mathcal{Y}|$-closeness holds. □

2-closeness. When extending f_1 to f_2, the 2-closeness property cannot be satisfied easily. The reason is that now f_{pre} maps a point $\notin \mathcal{X}$ into an arbitrary point. This point is then "rounded" to one of the two closest points in $[0, t-1, r]$ as for f_1, when evaluating f_{post}. The only exception is that when $r' = r$, f_{pre} simply shifts the inputs. In this case, $|\mathcal{Y}|$-closeness is improved to 2-closeness.

[16]Note that even if we do have an identity mapping (i.e., $\mathcal{Y} = [u, v, r']$), the f_{post} in the previous section is not enough as we need to revert the influence of f_{pre}.

[17]To briefly explain, the input $\vec{m}[i] \in [u, v, r']$, and thus $f_{\mathsf{pre}}(\vec{m}[i] \in [0, (v-u) \cdot r/r', r])$, and we have $(v-u) \cdot r/r' = t-1$. Therefore, we have $\lceil f_{\mathsf{pre}}(\vec{m}[i])/r \rfloor \cdot r = f_{\mathsf{pre}}(\vec{m}[i])$, and thus $f_{\mathsf{pre}}(\vec{m}[i]) = \vec{m}'[i]$.

Algorithm 2 BFV Bootstrapping for $f_2 : [u, v, r'] \to \mathcal{Y}$

```
1:  procedure Setup($1^\lambda$)
2:      Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of Boot below:
3:      (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
4:      (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate SlotToCoeff followed by any degree-1 polynomial function.
5:      (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating PartialDec followed by an arbitrary degree-$t$ polynomial function, still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.      ▷ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
6:      Let $r$ is the error bound such that $\Pr[|\mathsf{err}(\mathsf{sk}, \mathsf{ct}_3)| < r/2] \geq 1 - \mathsf{negl}(\lambda)$, where $\mathsf{ct}_3$ is in line 19 below.
7:      Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
8:      $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}})$
9:      $\mathcal{F}_2 := \{f_2(x) : [u, v, r'] \to \mathcal{Y} \mid (u, v, r' \in \mathbb{Z}_t) \wedge (\mathcal{Y} \subset \mathbb{Z}_t) \wedge (|\mathcal{Y}| \leq |[u, v, r']| = \frac{t-1}{r})\}$
10:     return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_2, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
11: procedure Boot($\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_2, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_2$)      ▷ For the sake of space, we call the GeneralFramework function defined in Algorithm 3.
12:     If $f_2 \notin \mathcal{F}_2$, abort.
13:     Let the input domain of $f_2$ be $[u, v, r]$.
14:     $f_{\mathsf{pre}}(x) := (x - u) \cdot r \cdot (r')^{-1}$
15:     $f_{\mathsf{post}}(x) = f_2(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$
16:     $\mathsf{ct}_1 \leftarrow \mathsf{BFV.Eval}(\mathsf{evk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pre}})$
17:     $\mathsf{ct}_2 \leftarrow \mathsf{ct}_1 \times U^\intercal$ (evaluated homomorphically)
18:                                 ▷ Recall that $U$ is defined in Section 4.1 and this step is SlotToCoeff
19:     $\mathsf{ct}_3 \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_2, t)$
20:     Parse $\mathsf{ct}_3 = (a_3 = \sum_{i\in[N]} a_3[i]X^{i-1}, b_2 = \sum_{i\in[N]} b_3[i]X^{i-1}) \in \mathcal{R}_t^2$
21:     Let $A_3 := \begin{pmatrix} a_3[1] & a_3[2] & a_3[3] & \dots & a_3[N] \\ -a_3[N] & a_3[1] & a_3[2] & \dots & a_3[N-1] \\ -a_3[N-1] & -a_3[N] & a_3[1] & \dots & a_3[N-2] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_3[2] & -a_3[3] & -a_3[4] & \dots & a_3[1] \end{pmatrix} \in \mathbb{Z}_t^{N\times N}$, and $\vec{b}_3 \leftarrow (b_3[i])_{i\in[N]} \in \mathbb{Z}_t^N$
22:     $\mathsf{ct}_4 \leftarrow \vec{b}_3 - \mathsf{ct}_{\mathsf{sk}} \times A_3$ (evaluated homomorphically)      ▷ i.e., PartialDec($\mathsf{ct}_3, \mathsf{sk}$)
23:     $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{BFV.Eval}(\mathsf{evk}, \mathsf{ct}_4, f_{\mathsf{post}})$
24:     return $\mathsf{ct}_{\mathsf{out}}$.
```

4.3 General Framework

We now introduce the general framework abstracted from the constructions we described above for the two function families. Looking ahead, the rest of the work highly relies on this framework, and only small local changes are made for different function families.

The general framework is straightforward, a visualization is provided in Fig. 1. It is formalized in Algorithm 3, where KeyGen is used to generate bootstrapping keys given f_{pre} and f_{post}; and GeneralFramework is used to formalize our general Boot procedure. Note that setting up the noise budgets is not included, since it is more function-family-dependent and we leave it to the setup algorithm for each function family. It is clear that both Algorithm 1 and Algorithm 2 can be modified to invoke the interfaces in Algorithm 3. In Section 5, all the constructions directly call the procedures in Algorithm 3.

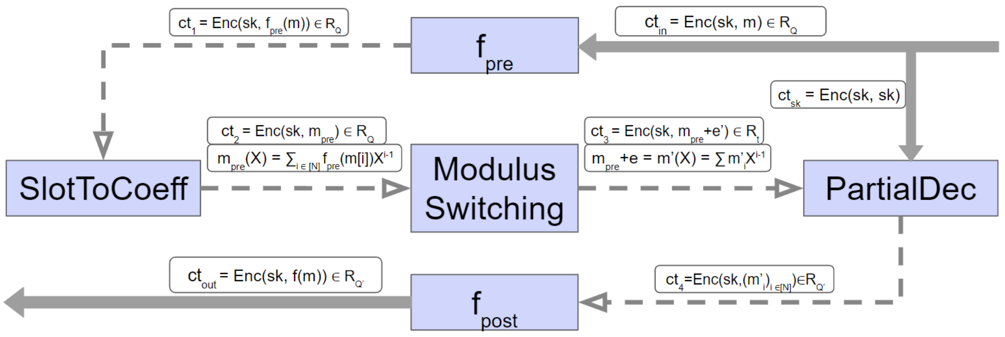
ct_1 = Enc(sk, f_{pre}(m)) ∈ R_Q
f_{pre}
ct_{in} = Enc(sk, m) ∈ R_Q
ct_{sk} = Enc(sk, sk)
ct_2 = Enc(sk, m_{pre}) ∈ R_Q
$m_{pre}(X) = \sum_{i \in [N]} f_{pre}(m[i])X^{i-1}$
ct_3 = Enc(sk, m_{pre}+e') ∈ R_t
$m_{pre}+e = m'(X) = \sum m'_i X^{i-1}$
SlotToCoeff
Modulus Switching
PartialDec
ct_{out} = Enc(sk, f(m)) ∈ $R_{Q'}$
f_{post}
ct_4=Enc(sk,$(m'_i)_{i \in [N]}$)∈$R_{Q'}$

Figure 1: The high-level illustration of our generalized framework

4.4 Optimizations

In this section, we specify some techniques that can be applied to our framework during implementation, which are orthogonal to our main procedure and do not affect correctness. We separate those techniques from our main framework for better readability.

More efficient linear transformation. In Algorithm 3, both line 11 and line 16 need to homomorphically evaluate a linear transformation for a vector-matrix multiplication. Doing this in a naive way requires N homomorphic multiplications and rotations. However, we can apply the baby-step-giant-step algorithm, introduced in [32] and later improved by [37], to reduce the computation to only $O(\sqrt{N})$ rotations (including $\sqrt{N}$ rotations of $\mathsf{ct}_{\mathsf{sk}}$, which can be saved with the technique below).

Key switching. Note that in Algorithm 3 line 16, the linear transformation is a length N vector sk multiplied by an $N \times N$ matrix U. However, by shortening the sk vector from N to $n \ll N$ via ring switching [28] (or field switching) technique, we are able to perform the linear transformation with simply n multiplications. On the other hand, to avoid violating security constraints, we first need to modulus switch ct_2 into a ciphertext ct'_2 with smaller modulus $Q' \ll Q$, and choose the smallest power-of-two n such that $\mathsf{RLWE}_{n,Q',\mathcal{D},\sigma}$ holds. Then, we generate another secret key $\mathsf{sk}' \in \mathcal{R}$ where only the first n coefficients of sk' are from $\mathcal{D}$ and the rest are padded with zeros. Afterwards, we perform a key switching to ct'_2 and generate another ciphertext ct''_2 such that $\mathsf{Dec}(\mathsf{sk}', \mathsf{ct}''_2) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}'_2) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_2)$. Since sk' has only n non-zero coefficients, line 16 can be replaced with a length-n-vector-by-$(n \times N)$-matrix multiplication, for some $n \ll N$.

Combining with the technique above, this requires only n multiplications and $\sqrt{n}$ rotations [37, Fig. 5].

Generating rotations in advance. Instead of doing those $\sqrt{n}$ rotations for ciphertext encrypting sk when evaluating Boot, since this step is independent of the input ciphertext, we can compute them during KeyGen in advance and include all the rotated keys in btk. This can save the $\sqrt{n}$ rotations during the bootstrapping phase.

NTT form preprocessing. To evaluate a ciphertext-by-plaintext multiplication homomorphically, one needs to (1) perform NTT transformations to both the ciphertext and the plaintext, (2) multiply the NTT results coefficient-wisely, and (3) apply an inverse-NTT (INTT) transformation to the result to finish the final evaluation. Since U in line 11 and $\mathsf{ct}_{\mathsf{sk}}$ in line 16 of Algorithm 3 are known in advance, we perform the step (1) NTT transform to ciphertext $\mathsf{ct}_{\mathsf{sk}}$ and plaintext U in advance during KeyGen. This, again, saves a lot of time during the online Boot phase.

Level-specific rotation keys. For line 16, since there have been a deep circuit evaluated prior to this step, we can first modulus switch the ciphertext down to $Q' \ll Q$ before applying rotations (still with enough noise

[18]Recall that in Section 2, we mentioned that our encryption includes an encoding procedure. Thus, the encrypted secret key is encoded as well. This means that we do not need an explicit encoding procedure (or homomorphic encoding procedure) as the counterpart of our decoding procedure SlotToCoeff, since it is accomplished by this line together with line 16 below.

Algorithm 3 General Framework

```
1: procedure $\mathsf{KeyGen}(1^\lambda, \mathsf{pp_{bfv}})$
2:     Prase $\mathsf{pp_{bfv}} = (N, Q, \mathcal{D}, \sigma, t)$.
3:     Generate BFV secret key $\mathsf{sk} = \sum_{i\in[N]} s_i X^{i-1} \leftarrow \mathcal{D}$, and let $\vec{s} := (s_i)_{i\in[N]} \in \mathbb{Z}_t^N$.
4:     Generate fresh encryption of BFV secret key $\mathsf{sk} \leftarrow \mathcal{D}$.
5:     Generate $\mathsf{ct_{sk}} \leftarrow \mathsf{Enc}(\mathsf{sk}, \vec{s})$ with $\mathsf{pp_{bfv}}$[18]
6:     Generate BFV public key $\mathsf{pk}$ and evaluation key $\mathsf{evk}$, using $\mathsf{sk}$.
7:     Let $\mathsf{btk} := (\mathsf{pk}, \mathsf{evk}, \mathsf{ct_{sk}})$.
8:     return $\mathsf{sk}, \mathsf{btk}$.
9: procedure $\mathsf{GeneralFramework}(\mathsf{pp}, \mathsf{btk} = (\mathsf{pk}, \mathsf{evk}, \mathsf{ct_{sk}}), \mathsf{ct_{in}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
10:    $\mathsf{ct}_1 \leftarrow \mathsf{BFV.Eval}(\mathsf{evk}, \mathsf{ct_{in}}, f_{\mathsf{pre}})$
11:    $\mathsf{ct}_2 \leftarrow \mathsf{ct}_1 \times U^\intercal$ (evaluated homomorphically)
12:                                  ▷ Recall that $U$ is defined in Section 4.1 and this step is $\mathsf{SlotToCoeff}$
13:    $\mathsf{ct}_3 \leftarrow \mathsf{ModSwitch}(\mathsf{ct}_2, t)$
14:    Parse $\mathsf{ct}_3 = (a_3 = \sum_{i\in[N]} a_3[i]X^{i-1}, b_2 = \sum_{i\in[N]} b_3[i]X^{i-1}) \in \mathcal{R}_t^2$
15:    Let $A_3 := \begin{pmatrix} a_3[1] & a_3[2] & a_3[3] & \dots & a_3[N] \\ -a_3[N] & a_3[1] & a_3[2] & \dots & a_3[N-1] \\ -a_3[N-1] & -a_3[N] & a_3[1] & \dots & a_3[N-2] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -a_3[2] & -a_3[3] & -a_3[4] & \dots & a_3[1] \end{pmatrix} \in \mathbb{Z}_t^{N\times N}$, and $\vec{b}_3 \leftarrow (b_3[i])_{i\in[N]} \in \mathbb{Z}_t^N$
16:    $\mathsf{ct}_4 \leftarrow \vec{b}_3 - \mathsf{ct_{sk}} \times A_3$ (evaluated homomorphically)      ▷ i.e., $\mathsf{PartialDec}(\mathsf{ct}_2, \mathsf{sk})$
17:    $\mathsf{ct_{out}} \leftarrow \mathsf{BFV.Eval}(\mathsf{evk}, \mathsf{ct}_4, f_{\mathsf{post}})$
18:    return $\mathsf{ct_{out}}$.
```

budget left to finish the rest of the process with $\mathcal{B}_{\mathsf{out}}$ noise budget left). This means that we can generate the rotation keys with modulus Q' instead of Q to reduce the size of the btk.

Combining SlotToCoeff and f_{pre}. In Algorithm 2, we have $f_{\mathsf{pre}} = (x - u) \cdot r \cdot (r')^{-1}$, where $x - u$ is a ciphertext and $r \cdot (r')^{-1}$ can be treated as a scalar. Thus this function involves a ciphertext-by-scalar multiplication and we combine this part with $\mathsf{SlotToCoeff}$ (which is a more complex ciphertext-by-scalar multiplication) to save one level of multiplication. To be more specific, we multiply the scalar with U in plaintext: $U' \leftarrow (r \cdot (r')^{-1})U$. Then, instead of homomorphically evaluating f_{pre}, we simply compute $\mathsf{ct}_1 \leftarrow \mathsf{ct}_1 - u$ followed by $\mathsf{ct}_2 \leftarrow \mathsf{ct}_1 \times U'$. This achieves exactly the same result as our current procedure but with one less level of multiplication.

The Paterson-Stockmeyer algorithm. Homomorphically evaluating a degree-d polynomial in a native way requires d scalar-by-ciphertext multiplications and d ciphertext-by-ciphertext multiplications. However, we apply Paterson-Stockmeyer [60] algorithm, and reduce the computation cost to d scalar-by-ciphertext multiplications and $O(\sqrt{d})$ ciphertext-by-ciphertext multiplications.

5 A More Fine-grained Construction

In this section, we make our construction's efficiency more fine-grained: i.e., dependent on the function families it needs to support.

Recall that for all the constructions introduced in Section 4, the circuits for evaluating f_{post}, which is of degree $O(t)$, have $O(t)$ multiplications. Notice that t needs to be at least $2N + 1$ to allow N slots for the best-amortized efficiency. In some applications, t needs to be even larger to allow a larger finite field computation. The efficiency is therefore greatly hampered. A natural question is can we make the number of multiplications and the degree of f_{post} $o(t)$?

Unfortunately, if we are mapping $O(t)$ of $\mathbb{Z}_t$ elements into $\mathbb{Z}_t$ elements (which is indeed the case for all the functions introduced above), we need a polynomial function of degree $\Omega(t)$. This makes it intuitively impossible to improve the efficiency asymptotically. Thus, in this section, we discuss some other function families with a more limited input domain (e.g., simply mapping $z \in \mathbb{Z}_t$ to $y \in \mathbb{Z}_t$ and $z' \neq z \in \mathbb{Z}_t$ to $y' \neq y \in \mathbb{Z}_t$), and show how to support them by evaluating a polynomial with a much smaller degree and thus provide better efficiency.

5.1 Point functions

5.1.1 Efficient Bootstrapping for Two-point Functions

We start with the simplest case: a function mapping two points to two points. We formalized this two-point mapping function as follows:

$$f_{\mathsf{2points}}(x) = \begin{cases} y & \text{if } x = z \\ y' & \text{if } x = z' \end{cases}$$

where $z \neq z', y \neq y' \in \mathbb{Z}_t$.[19]

We apply our generalized framework introduced in Section 4 by passing the correct $f_{\mathsf{pre}}, f_{\mathsf{post}}$ accordingly. Similar to the general issue discussed in Section 4.2, since $z \neq z'$ can be arbitrary, f_{pre} needs to be used to scale the intervals $|z - z'|$ by mapping z to v and z' to v' such that $|v - v'| \geq r$, where r again is the error bound. For simplicity, we choose $v = 0, v' = r$. Thus, $f_{\mathsf{pre}}(x) := r(x - z)(z' - z)^{-1}$. If $x = z$, $f_{\mathsf{pre}}(x) = 0$; if $x = z'$, $f_{\mathsf{pre}}(x) = r$.

Then, for f_{post}, we simply need to (homomorphically) map the ciphertext resulting from the partial decryption to y or y'. Formally:

$$f_{\mathsf{post}}(x) = \begin{cases} y & \text{if } x \in (-r/2, r/2) \\ y' & \text{if } x \in (r/2, 3r/2) \end{cases}$$

Note that this function only has $< 2r \ll t$ roots [20], which means that the degree of the function and the number of multiplications to evaluate this polynomial are both $O(r)$ instead of $O(t)$.

5.1.2 Extending to Multiple Points

Now we discuss how to extend this idea to support more than 2 points.

Revisiting Function Family $\mathcal{F}_2$. Let us first take a closer look at the function discussed in Section 4.2: $f_2 : [u, v, r'] \to \mathcal{Y} \in \mathcal{F}_2$, $u, v, r' \in \mathbb{Z}_t$, and $\mathcal{Y} \subset \mathbb{Z}_t$, and let $S := |[u, v, r']|$, we have $|\mathcal{Y}| \leq S \leq \frac{t-1}{r}$. The reason why the degree of f_{post} for f_2 needs to be $O(t)$ is that $S = \frac{t-1}{r}$, and by mapping all the inputs within error bound r to their corresponding outputs, we are eventually mapping all points $\in \mathbb{Z}_t$. In other words, if we make S strictly less than $\frac{t-1}{r}$, the pre-processing and post-processing can be evaluated by polynomials with degree $< t$.

To be more specific, denote f_2' to be this variant of f_2 with $S < \frac{t-1}{r}$. To perform functional bootstrapping for f_2', the preprocessing function remains unchanged: $f_{\mathsf{pre}}(x) := (x - u) \cdot r \cdot (r')^{-1}$, and set $f_{\mathsf{post}}(x) = f_2'(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$, for $x \in [-r/2, (S-1) \cdot r + r/2]$.

The difference between the bootstrapping procedure for f_2 and f_2' is that now f_{pre} has S roots and f_{post} has $S \cdot r$ roots, which largely reduces the degree and number of multiplications needed when S is small.

Multi-point mapping function family. The above high-level idea can be extended to a more general multi-point mapping function family $\mathcal{F}_{\mathsf{pts}} = \{f_{\mathsf{pts}} : \mathcal{X} \to \mathcal{Y},\ \mathcal{X}, \mathcal{Y} \subseteq \mathbb{Z}_t, |\mathcal{Y}| \leq |\mathcal{X}|\}$. Denote $S = |\mathcal{X}|$, and let $\mathcal{X} = \{x_1, \ldots, x_S\}$. Similarly, to map the input to $[0, (S-1) \cdot r, r]$. we define $f_{\mathsf{pre}}(m) := i \cdot r$ if $m = x_i$. Then, the post-processing function remains mostly the same: $f_{\mathsf{post}}(x) := f_{\mathsf{pts}}(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$, for $x \in [-r/2, (S-1) \cdot r + r/2]$.

By interpolating $f_{\mathsf{pre}}, f_{\mathsf{post}}$ of polynomials with degree $S, S \cdot r$ respectively, we have the construction for a general multi-point mapping function. We formally present our construction in Algorithm 4.

[19]Note that if $y = y'$, the construction is trivial. Simply return $\mathsf{Enc}(\mathsf{sk}, y)$ suffices, as the correctness definition does not explicitly define the behavior for $f(x)$ if $x \notin \{z, z'\}$.

[20]Recall that $r = O(\sqrt{h})$ where h is the hamming weight of the secret key.

Algorithm 4 BFV Bootstrapping for $f_{\mathsf{pts}} : \mathcal{X} \to \mathcal{Y}$

```
1: Let $S$ be some publicly known parameter, denoting the size of the input domain.
2: procedure $\mathsf{Setup}(1^\lambda)$
3:     Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of $\mathsf{Boot}$ below:
4:     (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
5:     (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate $\mathsf{SlotToCoeff}$ followed by any degree-$(S-1)$ polynomial function.
6:     (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating $\mathsf{PartialDec}$ followed by an arbitrary degree-$(S \cdot r - 1)$ polynomial function ($r$ defined below), still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.    $\triangleright$ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
7:     Let $r$ be the error bound such that error of $\mathsf{ct}_3$ in line 13 in Algorithm 3 $\leq r$ with probability $1 - \mathsf{negl}(\lambda)$
8:     Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
9:     $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
10:    $\mathcal{F}_{\mathsf{pts}} := \{\mathcal{X} \to \mathcal{Y} \mid \mathcal{X}, \mathcal{Y} \subset \mathbb{Z}_t, |\mathcal{Y}| \leq |\mathcal{X}| = S \leq \frac{t-1}{r}\}$
11:    return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{pts}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
12: procedure $\mathsf{Boot}(\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{pts}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pts}})$
13:    If $f_{\mathsf{pts}} \notin \mathcal{F}_{\mathsf{pts}}$, abort.
14:    Let the input domain of $f_{\mathsf{pts}}$ be $\mathcal{X} = \{x_1, \ldots, x_S\}$ (padded by dummy elements if less than $S$ elements).
15:    $f_{\mathsf{pre}}(m) := i \cdot r$ if $m = x_i$ (interpolated as a degree-$(S-1)$ polynomial)
16:    $f_{\mathsf{post}}(x) = f_{\mathsf{pts}}(f_{\mathsf{pre}}^{-1}(r \cdot \lceil x/r \rfloor))$, if $x \in (-r/2, (S-1) \cdot r + r/2)$ (interpolated as a degree-$(S \cdot r - 1)$ polynomial)
17:    $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{GeneralFramework}(\mathsf{pp}, \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
18:    return $\mathsf{ct}_{\mathsf{out}}$.
```

Theorem 5.1. Algorithm 4 is a correct BFV functional bootstrapping (Definition 3.1) procedure for function family $\mathcal{F}_{\mathsf{pts}} := \{f_{\mathsf{pts}} : \mathcal{X} \to \mathcal{Y} \mid \mathcal{X}, \mathcal{Y} \subset \mathbb{Z}_t, |\mathcal{Y}| \leq |\mathcal{X}| = S < \frac{t-1}{r}\}$, with t is from pp generated by Setup, assuming the correctness of BFV.

Proof. Given that f_{pre} has at most S roots and f_{post} has at most $S \cdot r$ roots, condition (2) on line 5 in Algorithm 4 allows one to evaluate f_{pre} followed by $\mathsf{SlotToCoeff}$ correctly, and condition (3) on line 6 allows one to evaluate the homomorphic decryption followed by f_{post} correctly with $\mathcal{B}_{\mathsf{out}}$ bits of noise left. The rest of proof is exactly the same as in Theorem 4.2. ☐

Remark 5.2. Notice that when $S \ll t$, i.e., the input set $\mathcal{X}$ is very sparse over $\mathbb{Z}_t$, a low degree (with degree $\ll S$) function may already be enough to map $\mathcal{X}$ to $\mathcal{X}'$ such that for all $i \neq j \in \mathcal{X}'$, $|i - j| > r$. After obtaining $\mathcal{X}'$, we simply set $f_{\mathsf{post}}(m) := f_{\mathsf{pts}}(x)$ for $m \in [f_{\mathsf{pre}}(x) - r/2, f_{\mathsf{pre}}(x) + r/2], x \in \mathcal{X}$ (thereby $f_{\mathsf{pre}}(x) \in \mathcal{X}'$). I.e., f_{post} checks every possible $x \in \mathcal{X}$, and if m is within $r/2$ points for any of the $f_{\mathsf{pre}}(x)$, return $f_{\mathsf{pts}}(x)$.

Furthermore, if $\forall i \neq j \in \mathcal{X}$, we have $|i - j| > r$, then there is no pre-processing needed to scale the intervals in between. See Section 6 for a concrete example. For simplicity, in the formal construction, we consider the worst-case scenario and treat the degree of f_{pre} to be S.

ℓ-closeness. For f_{pts}, ℓ-closeness does not hold for any ℓ unless $S = \frac{t-1}{r}$ which goes back to f_2. This is because the f_{post} domain now only covers a subset of $\mathbb{Z}_t$ while the invalid input may become any point in $\mathbb{Z}_t$ before computing f_{post}.

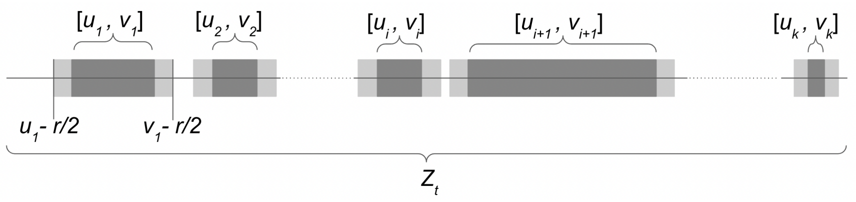

Figure 2: Depiction of the $\mathcal{X}$ of an example f_{ranges} over $\mathbb{Z}_t$.

5.2 Range Functions

Now instead of only allowing points, we focus on function mapping ranges (i.e., $[a, b]$ for $a < b \in \mathbb{Z}_t$) to points. Naively, we can treat a range $[a, b]$ simply as $b - a$ points, $a, b \in \mathbb{Z}_t$, and we reuse the scheme of the point-to-point functions above for range-to-point functions. However, this naive approach not only limits the efficiency but suffers from the constraints of having at most $\frac{t-1}{r}$ points across all ranges.

Fortunately, it turns out that if the ranges are well-separated, we could again construct a bootstrapping scheme with much better performance. Formally, define $\mathcal{F}_{\mathsf{ranges}} := \{f_{\mathsf{ranges}} : (\mathcal{X}_i)_{i\in[k]} \to \mathcal{Y}\}, \mathcal{X}_i \subset \mathbb{Z}_t, \mathcal{Y} = \{y_1, \ldots, y_k\} \subset \mathbb{Z}_t, k > 1$, and:

$$f_{\mathsf{ranges}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2 \\ \cdots & \\ y_k & \text{if } m \in \mathcal{X}_k \end{cases},$$

where $\mathcal{X}_i = [u_i, v_i], \forall i \in [k]$; furthermore, for all $i \neq j \in [k]$, $[u_i - r/2, v_i + r/2] \cap [u_j - r/2, v_j + r/2] = \emptyset$, where r is the error bound. Fig. 2 depicts a high-level view of an example f_{ranges} input $\mathcal{X}$.

Notice that for such a type of function, no pre-processing is needed and we safely set f_{pre} to be the identity function, and then set $f_{\mathsf{post}}(m) := y_i$ if $m \in (u_i - r/2, v_i + r/2), \forall i \in [k]$. Let $\mathcal{X} := \bigcup \mathcal{X}_{i\in[k]}$. Since f_{post} has $|\mathcal{X}| + k \cdot r$ roots [21], it has degree $|\mathcal{X}| + k \cdot r - 1$. We formalize our construction in Algorithm 5.

Theorem 5.3. Algorithm 5 is a correct BFV functional bootstrapping (Definition 3.1) procedure for function family defined on line 10, assuming the correctness of BFV.

Proof. Given that the underlying BFV is correct (i.e., all the homomorphic evaluations are completed as expected given enough noise budget), we have the following. Let $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct_{in}}) \in \mathbb{Z}_t^N$, and $\vec{m}_1 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) \in \mathbb{Z}_t^N$. Since f_{pre} is an identity function, $\vec{m}_1 = \vec{m}$. Then, let $ct_2 = (a_2, b_2) \in \mathcal{R}_Q^2$, $m_2 \leftarrow \lceil (t/Q)(b - a_2 \cdot \mathsf{sk}) \rfloor$, we have $m_2 = \sum_{i\in[N]} \vec{m}[i] X^{i-1} \in \mathcal{R}_t$, by condition (2) (that there is enough noise budget for f_{pre} and $\mathsf{SlotToCoeff}$). Let $\mathsf{ct}_3 = (a_3, b_3) \in \mathcal{R}_t^2$, $m_3 = \sum_{i\in[N]} m_3[i] X^{i-1} \leftarrow b_3 - a_3 \cdot \mathsf{sk} \in \mathcal{R}_t$, Then, it holds that $\Pr[m_3[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2)] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in [N]$, by condition (4) (that the error range r is large enough). Thus, let $\vec{m}_4 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_4) \in \mathbb{Z}_t^N$, for all $i \in [N]$, $\vec{m}_4[i] = m_3[i]$. Lastly, let $\vec{m}_5 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct_{out}}) \in \mathbb{Z}_t^N$, we have $\vec{m}_5[i] = y_j$ if $\vec{m}_4 \in (u_j - r/2, v_j - r/2) \forall j \in [N]$ by f_{post} and condition (5) (that there is enough noise budget to evaluate homomorphic decryption and f_{post}). Since we have $m_3[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2) \in (u_j - r/2, v_j - r/2)$ (given that $\vec{m}[i] \in [u_j, v_j]$ for some $j \in [k]$), we have $\vec{m}_5[i] = y_j$ for all $i \in [N]$ as expected. $\square$

Remark 5.4 (ℓ-closeness for f_{ranges})**.** While our construction does not naturally support 2-closeness for f_{ranges}, it can be achieved with some overhead. We modify f_{post} to be:

$$f_{\mathsf{post}}(m) = \begin{cases} y_i & \text{if } (u_i - r/2, v_i + r/2), \forall i \in [k] \\ f_{\mathsf{post}}(m') & \text{Otherwise} \end{cases},$$

[21]Technically speaking, since it is $(a_i - r/2, b_i + r/2)$, it only has $|\mathcal{X}| + k(r-1)$ roots. However, it is distracting to either make the range check non-symmetric (i.e., change to $(a_i - r/2, b_i + r/2]$) or calculate the number of roots more exactly (i.e., $k(r-1)$ instead of kr). Therefore, for here and also the rest of the paper, we estimate the number of roots roughly, the same way as in the main paper body now, for better readability.

Algorithm 5 BFV Bootstrapping for f_{ranges}

```
1: Let $S, k$ be two publicly known variables, where $S$ denotes the size of the input domain, and $k$ denotes the total number of ranges.
2: procedure $\mathsf{Setup}(1^\lambda)$
3:     Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of $\mathsf{Boot}$ below:
4:     (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
5:     (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate $\mathsf{SlotToCoeff}$ (since $f_{\mathsf{pre}}$ is identity function).
6:     (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating homomorphic decryption followed by an arbitrary degree-$(S + r \cdot k - 1)$ polynomial function, still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.    ▷ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
7:     Let $r$ be the error bound such that error of $\mathsf{ct}_3$ in line 13 in Algorithm 3 $\leq r$ with probability $1 - \mathsf{negl}(\lambda)$
8:     Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
9:     $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
10:    $\mathcal{F}_{\mathsf{ranges}} := \{f_{\mathsf{ranges}} \mid (f_{\mathsf{ranges}}$ with the following format$) \wedge (k > 1) \wedge (\mathcal{X}_i = [u_i, v_i] \subset \mathbb{Z}_t, [u_i - r/2, v_i + r/2] \cap [u_j - r/2, v_j + r/2] = \emptyset, \forall i \neq j \in [k])\}$
```

$$f_{\mathsf{ranges}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2 \\ \dots & \\ y_k & \text{if } m \in \mathcal{X}_k \end{cases}$$

```
11:    return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ranges}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
12: procedure $\mathsf{Boot}(\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ranges}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{ranges}})$
13:    If $f_{\mathsf{ranges}} \notin \mathcal{F}_{\mathsf{ranges}}$, abort.
14:    $f_{\mathsf{pre}}(m) := m$
15:    $f_{\mathsf{post}}(m) := y_i$ if $m \in (u_i - r/2, v_i + r/2), \forall i \in [k]$ (interpolated as a polynomial with degree at most $S + k \cdot r - 1$)
16:    $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{GeneralFramework}(\mathsf{pp}, \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
17:    return $\mathsf{ct}_{\mathsf{out}}$.
```

where $m' \in \mathcal{X}$ satisfying $m' - m \in \mathbb{Z}_t = \min_j(j - m), \forall j \in \mathcal{X}$. In this case, 2-closeness is straightforward (similar to the proof of Theorem 4.1). The overhead with this new f_{post} is then essentially $\frac{t}{|\mathcal{X}|+kr}$, which may be relatively insignificant (depending on the input function). The worst-case overhead is essentially bounded by $\frac{t}{2r}$ (and recall that evaluating f_{post} is only one component of the entire process).

Alternatively, Algorithm 7 in Section 5.2.2 provides an alternative way to evaluate f_{ranges} (with different efficiency tradeoffs) and provides k-closeness for free.

5.2.1 Two Unbalanced Ranges

If we have $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$ with two ranges only, denote $S_1 = |\mathcal{X}_1|, S_2 = |\mathcal{X}_2|$, the method above would need a degree-$(S_1 + S_2 + 2r - 1)$ function with $S_1 + S_2 + 2r$ multiplications. However, if we assume that the sizes of these two ranges are extremely unbalanced, w.l.o.g., $S_2 \gg S_1$, we are able to further reduce the computation work down to $S_1 + r + \log(t) + 1$ multiplications, which can be much more efficient. Formally, we define: $\mathcal{F}_{\mathsf{ub}} := \{f_{\mathsf{ub}} : \mathcal{X}_1, \mathcal{X}_2 \to y_1, y_2\}, \mathcal{X}_{i\in[2]} \subset \mathbb{Z}_t, y_{i\in[2]} \in \mathbb{Z}_t$, such that:

$$f_{\mathsf{ub}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2 \end{cases}$$

where $\mathcal{X}_i = [u_i, v_i], |\mathcal{X}| = S_i$ for $i \in [2]$ and $S_1 \ll S_2$.

Regarding the more detailed construction, we again set f_{pre} to be the identity function, but use a new post-processing function $f_{\mathsf{post}}(m) := (\prod_{i\in(u_1-r/2,v_1+r/2)}(m-i))^{t-1}\cdot(y_2-y_1)+y_1$. The correctness analysis is as follows: recall that after homomorphic decryption, input $m[i] \in \mathcal{X}_1 = [u_1, v_1]$ is mapped to $(u_1-r/2, v_1+r/2)$. Thus, $c \leftarrow \prod_{i\in[u_1-r/2,v_1+r/2]}(m-i) = 0$, if $m[i] \in [u_1, v_1]$ and $c \in \mathbb{Z}_t, c \neq 0$ for $m[i] \notin \mathcal{X}_1$. By Fermat's Little Theorem, raising a non-zero field element up to $t-1$ would result in 1, i.e., $c^{t-1} = 1$ for all $c \in \mathbb{Z}_t, c \neq 0$. Therefore, $c^{t-1} = 0$ if $m[i] \in [u_1, v_1]$ and 1 otherwise. Lastly, we evaluate $c^{t-1}\cdot(y_2-y_1)+y_1$ so that the result would be y_1 if $m[i] \in \mathcal{X}_1$ and y_2 otherwise.

It is not hard to see that f_{post}, though with degree $(S_1+r-1)\cdot t+1$, only needs $S_1+r+\log(t)+1$ multiplications for evaluation.

We formalize our construction in Algorithm 6.

Algorithm 6 BFV Bootstrapping for function f_{ub}

```
1: Let $S_1$ be a publicly known parameter.
2: procedure Setup($1^\lambda$)
3:     Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of Boot below:
4:     (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
5:     (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate SlotToCoeff (since $f_{\mathsf{pre}}$ is identity function).
6:     (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating homomorphic decryption followed by an arbitrary polynomial with degree $(S_1+r)t$, still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.    ▷ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
7:     Let $r$ be the error bound such that error of $\mathsf{ct}_3$ in line 13 in Algorithm 3 $\leq r$ with probability $1 - \mathsf{negl}(\lambda)$
8:     Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
9:     $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
10:    $\mathcal{F}_{\mathsf{ub}} := \{f_{\mathsf{ub}} \mid (f_{\mathsf{ub}}$ with the following format$) \wedge |\mathcal{X}_1| \ll |\mathcal{X}_2| \wedge (\mathcal{X}_{i\in[2]} \in \mathbb{Z}_t) \wedge (\mathcal{X}_i = [u_i, v_i], [u_1 - r/2, v_1 + r/2] \cap [u_2 - r/2, v_2 + r/2] = \emptyset)\}$
           $$f_{\mathsf{ub}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2 \end{cases}$$
11:        return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ub}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
12: procedure Boot($\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ub}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{ub}}$)
13:     If $f_{\mathsf{ub}} \notin \mathcal{F}_{\mathsf{ub}}$, abort.
14:     $f_{\mathsf{pre}}(m) := m$
15:     $f_{\mathsf{post}}(m) := (\prod_{x\in[u_1-r/2,v_1+r/2]}(m-x))^{t-1}\cdot(y_2-y_1)+y_1$
16:     $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{GeneralFramework}(\mathsf{pp}, \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
17:     return $\mathsf{ct}_{\mathsf{out}}$.
```

Theorem 5.5. Algorithm 6 is a correct BFV functional bootstrapping (Definition 3.1) procedure for the function family defined on line 10, assuming the correctness of BFV. Furthermore, it is 2-close (Definition 3.2).

Proof. Given that the underlying BFV is correct (i.e., all the homomorphic evaluations are completed as expected given enough noise budget), we have the following. Let $\vec{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{in}}) \in \mathbb{Z}_t^N$, and $\vec{m}_1 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1) \in \mathbb{Z}_t^N$. Since f_{pre} is an identity function, $\vec{m}_1 = \vec{m}$. Then, let $\mathsf{ct}_2 = (a_2, b_2) \in \mathcal{R}_Q^2$, $m_2 \leftarrow \lceil (t/Q)(b - a_2\cdot\mathsf{sk})\rfloor$, we have $m_2 = \sum_{i\in[N]} \vec{m}[i]X^{i-1} \in \mathcal{R}_t$, by condition (2) (that there is enough noise budget for f_{pre} and SlotToCoeff). Let $\mathsf{ct}_3 = (a_3, b_3) \in \mathcal{R}_t^2$, $m_3 = \sum_{i\in[N]} m_3[i]X^{i-1} \leftarrow b_3 - a_3\cdot\mathsf{sk} \in \mathcal{R}_t$, Then, it holds that $\Pr[m_3[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2)] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in [N]$, by condition (4) (that the error range r is large enough). Thus, let $\vec{m}_4 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_4) \in \mathbb{Z}_t^N$, for all $i \in [N]$, $\vec{m}_4[i] = m_3[i]$. Lastly, let $\vec{m}_5 \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) \in \mathbb{Z}_t^N$, we have $\vec{m}_5[j] = ((\prod_{i \in [v_1 - r/2, u_1 + r/2]} (m_3[j] - i))^{t-1} \cdot (y_2 - y_1)) + y_1$ by the correctness of f_{post} (where $[v_1, u_1]$ denotes the first range) and condition (5) (that there is enough noise budget to evaluate homomorphic decryption and f_{post}, which has degree $\leq (v_1 - u_1 + r)t$). Since we have $m_3[i] \in (\vec{m}[i] - r/2, \vec{m}[i] + r/2) \in (a_j - r/2, b_j - r/2)$ (for some $j \in [2]$), we have $\vec{m}_5[i] = y_j$ for all $i \in [N]$ as expected.

The 2-closeness is easy to show. Since f_{post} covers all the values in $\mathbb{Z}_t$ and has only two possible outputs, if the input $\vec{m}[i]$ is invalid, it can also only output one of the two. Furthermore, even if $\vec{m}[i] \notin \mathcal{X}$, $\vec{m}_4[i] \in \mathbb{Z}_t$ simply because all the calculations are done using BFV. Thus, the output $\vec{m}_5[i] \in \mathcal{Y}$, which has only two options. □

Remark 5.6. For this construction, essentially, instead of treating the input domain as $\mathcal{X}_1, \mathcal{X}_2$, we are treating it as $\mathcal{X}_1 = [u_1, v_1], \mathcal{X}_2' := \mathbb{Z}_t \setminus [u_1 - r, v_1 + r]$. In other words, the actual function we evaluate is as follows:

$$f'_{\mathsf{ub}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2' \end{cases} .$$

Since there are only two possible results, 2-closeness is satisfied for free.

5.2.2 Generalized Unbalanced Ranges

Moreover, we can extend this two-unbalanced-ranges setting to multiple ranges, when one of them still has a dominant size. Formally, for k ranges $\mathcal{X}_1, \ldots \mathcal{X}_k$ that are well separated, let $\mathcal{X}_i = [u_i, v_i]$, $S_i = |\mathcal{X}_i|$, w.l.o.g., we assume $S_k > \sum_{i=1}^{k-1} S_i$. By applying a similar way to evaluate this type of function, we only need $2\left(\sum_{i=1}^{k-1} S_i + r(k-1)\right) + \log(t)$ multiplications, instead of $\sum_{i=1}^{k} S_i$. For S_k much larger than $\sum_{i=1}^{k-1} S_i$, this evaluation might be more efficient. Notice that this generalized unbalanced case evaluates the exact same function as f_{ranges} defined in Section 5.2, but we utilize its "unbalanced" property and thus evaluate it with an alternative method.

The construction is as follows. Let $\mathcal{X}_i' := (u_i - r/2, v_i + r/2)$ for all $i \in [k]$. Again f_{pre} is the identity function. To set f_{post}, we first define:

$$h(m) := \begin{cases} y_1 - y_k & \text{if } m \in \mathcal{X}_1' \\ y_2 - y_k & \text{if } m \in \mathcal{X}_2' \\ \cdots & \\ y_{k-1} - y_k & \text{if } m \in \mathcal{X}_{k-1}' \end{cases}$$

and $h(m)$ has degree $\sum_{i=1}^{k-1}(S_i + r)$. Then, we define

$$g(m) := \prod_{j \in \mathcal{X}_i', i \in [k-1]} (x - j)$$

$g(m)$ also has degree $\sum_{i=1}^{k-1}(S_i + r)$. Lastly, we define $f_{\mathsf{post}}(m) := h(m) \cdot (1 - g(m)^{t-1}) + y_k$. [22]

For $m \in \mathcal{X}_i'$ for all $i \in [k-1]$, $h(m) = y_i - y_k$, and $g(m) = 0$. Therefore, $h(m) \cdot (1 - g(m)^{t-1}) + y_k = (y_i - y_k) \cdot (1 - 0^{t-1}) + y_k = y_i$ as expected. On the other hand, if $m \in \mathcal{X}_k'$, $g(m) \neq 0$. Therefore, $h(m) \cdot (1 - g(m)^{t-1}) + y_k = h(m) \cdot (1 - 1) + y_k = y_k$.

With regard to the efficiency, both h, g requires $2\sum_{i=1}^{k-1}(S_i + r)$ multiplications, and therefore, in total, f_{post} requires $2\sum_{i=1}^{k-1}(S_i + r) + \log(t) + 1$ multiplications. On the other hand, the degree of the function is $(\sum_{i=1}^{k-1}(S_i + r))(t-1)$.

[22] Notice that f_{ub} is a special case of this construction, as for f_{ub} in Section 5.2.1, $h(m) = y_1 - y_2$ is simply a constant function and thus does not require any evaluation. Therefore, the cost is $S_1 + r + \log(t)$ for f_{ub} instead of $2S_1 + 2r + \log(t)$.

At first glance, this generalized unbalanced function might seem to be too subtle to be widely used. However, we believe that in some cases, it may still be helpful. For example, suppose we want to approximate a ReLU function: $r(x) = x$ if $x \in (0, t/4)$ and $r(x) = 0$ otherwise. Then, we can divide $(0, t/4)$ into $k - 1$ ranges, and use $\mathbb{Z}_t \setminus (0, t/4)$ as the k-th range, which has the dominant size. The larger k is, the finer divisions we have for the input sets, and thus the more accurate this approximation is. On the other hand, the k-th range is absolutely much larger than the other ones even combined, which can be evaluated efficiently using the construction we introduce above. We formalize our construction in Algorithm 7.

Algorithm 7 An alternative construction of BFV Bootstrapping for f_{ranges}

```
1: Let $S', k$ be two publicly known variables, where $S'$ denotes the size of the input domain except for the $k$-th range, and $k$ denotes the total number of ranges.
2: procedure Setup($1^\lambda$)
3:     Select $(N, Q, \mathcal{D}, \sigma, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, t)$ satisfying the following while minimizing the overall computation cost of Boot below:
4:     (1) $\mathsf{RLWE}_{N,Q,\mathcal{D},\chi_\sigma}$ holds.
5:     (2) Select the minimum $\mathcal{B}_{\mathsf{in}}$ such that a BFV ciphertext with ring dimension $N$, plaintext space $t$, and noise budget $\mathcal{B}_{\mathsf{in}}$, is enough to evaluate SlotToCoeff (since $f_{\mathsf{pre}}$ is identity function).
6:     (3) Select the minimum $Q$ such that a fresh BFV ciphertext with ring dimension $N$, plaintext space $t$, and ciphertext space $Q$, after evaluating homomorphic decryption followed by an arbitrary degree-$(S' + r(k-1))(t-1)$ polynomial function, still has $\mathcal{B}_{\mathsf{out}} = \mathcal{B}_{\mathsf{in}} + 1$ noise budget remaining.   ▷ $\mathcal{B}_{\mathsf{out}}$ can be replaced by any number dependent on applications.
7:     Let $r$ be the error bound such that error of $\mathsf{ct}_3$ in line 13 in Algorithm 3 $\leq r$ with probability $1 - \mathsf{negl}(\lambda)$
8:     Let $\mathsf{pp}_{\mathsf{bfv}} := (N, Q, \mathcal{D}, \sigma, t)$.
9:     $\mathsf{sk}, \mathsf{btk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{pp}_{\mathsf{bfv}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
10:    $\mathcal{F}_{\mathsf{ranges}} := \{f_{\mathsf{ranges}} \mid (f_{\mathsf{ranges}}$ with the following format$) \wedge (k > 1) \wedge (\mathcal{X}_i = [u_i, v_i] \subset \mathbb{Z}_t, [u_i - r/2, v_i + r/2] \cap [u_j - r/2, v_j + r/2] = \emptyset, \forall i \neq j \in [k])\}$
$$f_{\mathsf{ranges}}(m) = \begin{cases} y_1 & \text{if } m \in \mathcal{X}_1 \\ y_2 & \text{if } m \in \mathcal{X}_2 \\ \cdots \\ y_k & \text{if } m \in \mathcal{X}_k \end{cases}$$
11:        return $\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ranges}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{sk}, \mathsf{btk})$.
12: procedure Boot($\mathsf{pp} = (N, t, \mathcal{B}_{\mathsf{in}}, \mathcal{B}_{\mathsf{out}}, \mathcal{F}_{\mathsf{ranges}}, \mathsf{pp}_{\mathsf{aux}} = r), \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{ranges}}$)
13:    If $f_{\mathsf{ranges}} \notin \mathcal{F}_{\mathsf{ranges}}$, abort.
14:    $f_{\mathsf{pre}}(m) := m$
15:    Define
$$h(m) := \begin{cases} y_1 - y_k & \text{if } m \in (u_1 - r/2, v_1 + r/2) \\ y_2 - y_k & \text{if } m \in (u_2 - r/2, v_2 + r/2) \\ \cdots \\ y_{k-1} - y_k & \text{if } m \in (u_{k-1} - r/2, v_{k-1} + r/2) \end{cases}$$
       and
$$g(m) := \prod_{j \in \mathcal{X}'_i, i \in [k-1]} (x - j)$$
16:    $f_{\mathsf{post}}(m) := h(m) \cdot (1 - g(m)^{t-1}) + y_k$
17:    $\mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{GeneralFramework}(\mathsf{pp}, \mathsf{btk}, \mathsf{ct}_{\mathsf{in}}, f_{\mathsf{pre}}, f_{\mathsf{post}})$
18:    return $\mathsf{ct}_{\mathsf{out}}$.
```

Theorem 5.7. Algorithm 7 is a correct BFV functional bootstrapping (Definition 3.1) procedure for function family defined on line 10, assuming the correctness of BFV. Furthermore, it is k-close (Definition 3.2).

6 Evaluation

We implemented our algorithms proposed above in a C++ library, based on the SEAL [61] library. We benchmark these schemes on several parameter settings on a Google Compute Cloud `N4-standard-4` with 16GB RAM.

6.1 Performance of Our Construction

Parameter selection. We choose BFV parameters as follows: $N = 32768, t = 65537, \sigma = 3.2$. We use ternary secret keys with a hamming weight of 512.[23] The ciphertext modulus Q is chosen according to each function as specified in Table 2. These parameters guarantee > 128-bit security by LWE estimator [2] for all the function families we have tested (except for f_{ub}, which provides 106-bit of security, but for better comparison, we remain N, t, σ unchanged but reduce the security). To guarantee that the modulus switching error is bounded by r except with 2^{-40} probability,[24][25] we choose $r = 128$ (thus $r/2 = 64$).

We benchmark all the functions we described, including f_1 (i.e., the identity function over $[0, 65408 = t-1-r, 128 = r]$) in Algorithm 1; f_2 (i.e., mapping each point in $[a, b, r']$ to an arbitrary point $y \in \mathbb{Z}_t$) in Algorithm 2; point functions f_{pts} (i.e., mapping several points to several points) in Algorithm 4; range functions f_{ranges} (i.e., mapping several ranges to several points) in Algorithm 5; and unbalanced range functions f_{ub} (i.e., mapping two unbalanced ranges to two points) in Algorithm 6.

As r is fixed, the input of f_2 can be at most $\frac{t-1}{r} = 512$ points. Therefore, we choose $f_2 : [0, 1022, 2] \to \mathcal{Y}$ where $\mathcal{Y}$ is a random subset of $\mathbb{Z}_{65537}$ with 512 points. For $f_{\mathsf{pts}} : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} \subset \mathbb{Z}_{65537}$, we choose two different functions: the first function maps $\{0, 32768\}$ to two different random points, i.e., $\mathcal{X}$ and $\mathcal{Y}$ only contain two points, and thus achieve the best possible performance; the second one demonstrates a more general functionality by mapping eight random points to eight random points. For f_{ranges}, we choose two well-separated ranges each containing 127 points. For f_{ub}, we choose two very unbalanced ranges, one of which is of size $r-1$ and the other being $t - 2r + 1$.

For all these functions, we choose the smallest Q such that the output noise budget has $\sim$180 bits remaining. Thus, for all but f_{ub}, we can guarantee 128-bit security with our parameter, and for f_{ub}, we have about 106-bit security.

Performance analysis. As shown in Table 2, our amortized runtime is about 1-2 orders of magnitude faster than regular BFV bootstrapping: both for the runtime per slot and the runtime per effective bit (i.e., the runtime per slot divided by the effective input plaintext space in bits). Our functionality is slightly different from prior works: we only support correctness over a subset of the plaintext space, but we also allow a look-up table evaluation.

[59] has the plaintext space to be much smaller than the other regular bootstrapping constructions because enlarging the plaintext space requires some non-trivial modification to their construction. Therefore, they

[23]Our construction replies on sparse keys in the same way as prior works. We can extend our key to be uniform, but r needs to be increased accordingly, since $r = O(\sqrt{h})$ for h being the hamming weight.

[24]We choose security parameter $\delta = 40$ which is the same as in [54], since the error probability is statistical, and 40 is a relatively popular and reasonable statistical security parameter. Prior works in BGV/BFV bootstrapping instead choose error probability via evaluation: based on our private communication with the authors of the prior works, it was chosen such that no overflow happens during benchmarking tests. To our knowledge, other BFV bootstrapping works do not explicitly discuss how they choose the concrete error probability in the paper with concrete numbers, and thus we follow the parameter in [54]. Asymptotically, $r = O(\sqrt{\delta})$ when fixing other parameters.

[25]According to [15], 2^{-40} gives $\sim$50-bit of security for IND-CPA-D (introduced in [49]). To achieve 128-bit security of IND-CPA-D, roughly a failure probability of 2^{-120} is needed. To accommodate this, our error range grows from 128 to $\sim$216 and thus the effective plaintext space (for f_1, f_2) is reduced from 512 points to $\sim$302 points (and correspondingly other function families). Thus, our amortized per bit runtime would be just slightly increased. Furthermore, note that adjusting the IND-CPA-D security level would also affect the runtime in all prior works as well, which will thus maintain our advantage, if not further increase.

Function Family	Input Domain	# of slots	Ciphertext Modulus	Output Noise Budget	Total Runtime (sec)	**Runtime per slot (ms)**	**Runtime per bit (ms)**
Identity function f_1 over $[0, t-1-r, r]$, *Algorithm* 1	$[0, 65536, 128]$	32768	830	181	142.5	**4.35**	**0.48**
$f_2 : [u, v, r'] \to \mathcal{Y}$ $u, v, r' \in \mathbb{Z}_t$, $\mathcal{Y} \subset \mathbb{Z}_t$, *Algorithm* 2	$[0, 1022, 2]$				142.4	**4.34**	**0.48**
$f_{\mathsf{pts}_1} : \mathcal{X} \to \mathcal{Y}$, $\mathcal{X}, \mathcal{Y} \subset \mathbb{Z}_t$, $\lvert\mathcal{X}\rvert = \lvert\mathcal{Y}\rvert = 2$, *Algorithm* 4	$\{0, 32768\}$		590	198	18.7	**0.57**	**0.57**
$f_{\mathsf{pts}_2} : \mathcal{X} \to \mathcal{Y}$, $\mathcal{X}, \mathcal{Y} \subset \mathbb{Z}_t$, $\lvert\mathcal{X}\rvert = \lvert\mathcal{Y}\rvert = 8$, *Algorithm* 4, without pre-scale on $\mathcal{X}$	$\{57004, 46969, 21931, 39030, 59092, 9965, 30013, 58301\}$		650	194	24.8	**0.76**	**0.25**
$f_{\mathsf{pts}_3} : \mathcal{X} \to \mathcal{Y}$, $\mathcal{X}, \mathcal{Y} \subset \mathbb{Z}_t$, $\lvert\mathcal{X}\rvert = \lvert\mathcal{Y}\rvert = 8$, *Algorithm* 4 with pre-scale on $\mathcal{X}$				181	26.3	**0.80**	**0.27**
$f_{\mathsf{ranges}}(m) = y_i$ if $m \in [u_i, v_i]$, $u_i, v_i, y_i \in \mathbb{Z}_t, i \in [k], k \geq 2$, *Algorithm* 5	Two ranges: $[-63, 63]$ & $[32704, 32831]$		630	205	22.5	**0.69**	**0.09**
$f_{\mathsf{ub}}(m) = y_i$ if $m \in [u_i, v_i]$, $u_i, v_i, y_i \in \mathbb{Z}_t, i \in [2]$, *Algorithm* 6	Two ranges: $[-63, 63]$ & $\mathbb{Z}_{65537} \setminus [-127, 127]$		1070	180	34.3	**1.04**	**0.07**

Regular BFV bootstrapping [59] 128-bit security	$\mathbb{Z}_{257}$	128	881	507	22.0	**173.0**	**21.62**
Regular BFV bootstrapping [25] 66-bit security	$\mathbb{Z}_{127^2}$	2268	1134	330	95.0	**42.0**	**3.00**
Regular BFV bootstrapping [14] 126-bit security	$\mathbb{Z}_{257^2}$	128	806	245	42.0	**328.0**	**20.50**

Table 2: Batched bootstrapping for binary gates using our technique compared to the unoptimized construction in [54]. Notice that based on the BFV parameter we choose, all our constructions guarantee $>$ 128-bit security except for f_{ub} which is of 106-bit security; all our constructions are evaluated on input with 35-bit noise budget, except for f_{pts_3} which needs input with 125-bit noise budget. See Section 6 "Parameter selection" for details. The runtimes of prior works are taken directly from their papers. We use a basic GCP instance which does not grant us extra advantage over the runtime.

also have a relatively limited input domain (containing only 257 points). Among all of the regular BFV bootstrapping works, [25] provides the best performance but with a relatively low security guarantee (only 66 bits). To guarantee $>$ 100 bit security, their performance will be further reduced. As mentioned in [59], the techniques in [59] and [25] might be combined to achieve a better regular BFV bootstrapping construction, but it is still unlikely to be comparable with our constructions (again, they provide a stronger functionality by considering all values in the plaintext space as valid inputs). The main reason we outperform the regular bootstrapping framework by around 1 to 2 orders of magnitude, is that we make full use of 32768 slots per ciphertext.

f_1 and f_2 have roughly the same runtime and the same input noise budget requirement, as the evaluation of f_{pre} is combined with the $\mathsf{SlotToCoeff}$ step (recall that for f_2, f_{pre} is simply a degree-1 function) They both evaluate over 512 different points, thus requiring f_{post} to have degree $t-2$ (as $512 \cdot r = t - 1 = 65536$). Therefore, they are both the slowest among all the different types of functions. Also, as discussed, these two types of functions can be viewed as special cases of f_{pts}.

For f_{pts}, we test a function for two points, for which is the most efficient non-trivial function our protocol works. Such a function takes only about 1.5ms per slot. To show more generality, we also test functions with 8 points. All the points are randomly chosen. For the points we randomly chose, no f_{pre} is needed as they are all separated by at least $r = 128$. In this case, the runtime is only slightly slower than f_{pts} with two points. However, to show the worst case, a function of degree 7 is needed as the preprocessing function. We also benchmark it to show the difference. In this case, f_{pre} is a degree 7 function and therefore requires the input noise budget to be 90 bits more. The runtime is roughly the same. Note that, however, for such a small number of points (e.g., 8 points), it is more likely that f_{pre} does not need any preprocessing (or at least only a lower degree function like degree-one is needed, thus introducing little overhead, if any).

Then, we switch our focus to the range functions. Note that for f_{ranges}, each range contains 127 points. Therefore, there are a total of 254 points. However, the runtime is even faster than f_{pts} with only 8 points. This is because it only requires a degree-$254 + 2r = 510$ postprocessing function; in contrast f_{pts} with 8 points already requires the postprocessing function to have degree $8 \cdot r = 1024$. For the unbalanced ranges, we use

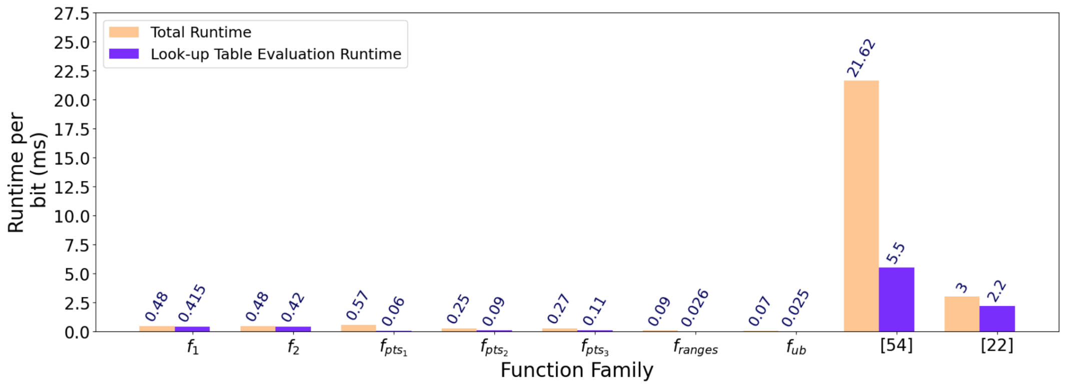

Figure 3: Bar chart illustration of total runtime and look-up table evaluation time per bit. Note that [25] provides only 66-bit of security while our construction and [59] both provide about 128-bit security.

$[-63, 63]$ and $\mathbb{Z}_t \setminus [-127, 127]$, containing 65409 points. However, it is also easy to see that the runtime is only slightly slower than the range function with two small ranges. The only drawback is that $\log(Q)$ is required to be very large since f_{post} is a function of degree $\sim 2r \cdot t$.

Runtime breakdown. As shown in Fig. 3, for some functions like f_1, f_2, the look-up table evaluation takes the majority of time, as their f_{post} has a much higher degree than the other ones. However, for all of the functions, our runtime is still greatly better than both of the prior works (and their breakdown is taken from the papers).

6.2 Performance Comparisons with Other Works

Comaprison with [41]. As mentioned in Section 1.2, there is a very recent concurrent and independent work that uses CKKS to bootstrap BFV with comparable performance (see Section 1.2 for a high-level pros and cons comparison). Here we compare our concrete performance with them.

Using a machine similar to their setup (GCP instance N4 with CPU Intel Emerald Rapids with 16 GB RAM, similar to the machine with CPU Intel Xeon Gold 6242 and 503 GB RAM in [41]), for some functions, we have faster runtime (e.g., $f_{\mathsf{pts}}, f_{\mathsf{ranges}}$) while others are worse. Furthermore, we support a function evaluation for free. For these reasons, we believe our performance is comparable and can be preferable in some scenarios.

In addition, we have another advantage in terms of scalability with respect to the noise budget after bootstrapping. In [41], the runtime is essentially $\sim (B/128)$ms[26] per slot where B is the noise budget after bootstrapping. However, our runtime grows more slowly in most cases. Therefore, for a larger noise budget, our runtime has more advantages.

Comparison with [56]. Another concurrent and independent work also improves bootstrapping for BFV (see Section 1.2 for high-level pros and cons comparisons). Concretely, with $t = 65537$, they achieve their best-amortized runtime to be $60/2784 \approx 20$ms per slot (with 80-bit security while we support 128-bit security). In comparison, our slowest function (f_1) is roughly 4.35 ms per slot (on GCP instance N4 with CPU Intel Emerald Rapids with 16 GB RAM, which is comparable to, if not slower than, their workstation with Intel i9-10980XE). Furthermore, we support functional bootstrapping.

Comparison with [43]. One more concurrent and independent work recently similarly focuses on BFV functional bootstrapping. In their benchmarks (parameter set I), they choose $q = 17$ (q serves as a similar functionality as our $\mathcal{Y}$), which is the output plaintext modulus (about 3-4 bits), and $p = 700$ (p serves as a similar functionality as our $\mathcal{X}$) which is the input plaintext modulus. They use such a parameter set as their concrete efficiency is linear in q (the efficiency also depends on r for some $r > 0$ such that $q^r > p$). Their

[26]This number is based on [41, Tab.2&Fig.6] together with the private communication with the authors.

runtime for this parameter set is about 46.5 seconds (on Intel(R) Xeon(R) Platinum 8268 @ 2.90GHz CPU and 192GB RAM) and their number of slots is similar to prior works in Table 2, about hundreds to a few thousand slots. This means that their amortized runtime is also about 1-2 orders of magnitude slower than our construction. A similar conclusion can be drawn for their parameter set II.

Comparison with other FHE bootstrapping. Since it is hard to directly compare to other FHE bootstrapping schemes concretely (as distinct schemes differ a lot in terms of settings), we give a brief high-level discussion. For CKKS, functional bootstrapping is particularly challenging as during bootstrapping, a polynomial approximation of the sine function is used. To capture functional bootstrapping, a polynomial approximation of that function together with sine needs to be done, which is very inefficient. Therefore, to the best of our knowledge, CKKS cannot easily support (even relaxed) functional bootstrapping. On the other hand, for FHEW/TFHE, while they natively support functional bootstrapping, they do not naturally support additions and multiplications before or after bootstrapping. Therefore, our method provides more flexibility (while the performance can be comparable when $|\mathcal{X}| = p$ where p is the plaintext space of FHEW/TFHE, based on our estimation with the numbers in [54] and Section 8).

7 Applications

In this section, we discuss some applications that can take advantage of our constructions. We first discuss one application in very recent work in detail and compare the result using our scheme with the results using prior works. We then introduce some potential applications at a high level.

7.1 Oblivious Permutation via BFV

A recent work [23] proposes a way to homomorphically permute a database with N entries. Essentially, it allows the server to obliviously permute a database using BFV such that the decrypted result is indistinguishable from a randomly permuted database. The permutation randomness comes from the client, but it takes only $o(N)$ communication: the server first uses a BFV-encrypted seed provided by the client and a BFV-friendly PRG to generate $O(N\log(N))$ random bits homomorphically; then, it homomorphically evaluates the Thorp shuffle (or equivalently, a butterfly shuffle) using these random bits. The Thorp shuffle consists of $h = O(\lambda)$ consecutive rounds, where each round divides the database into $N/2$ pairs of entries and then swaps each pair using a random bit. For example, if $N = 2$, the swap operation homomorphically computes $\mathsf{DB}'[1] \leftarrow \mathsf{DB}[1]\cdot r + \mathsf{DB}[2]\cdot(r-1)$ and $\mathsf{DB}'[2] = \mathsf{DB}[2]\cdot r + \mathsf{DB}[1]\cdot(r-1)$. If $r = 1$, $\mathsf{DB}' = \mathsf{DB}$, and $\mathsf{DB}' = \mathsf{DB}[2]\|\mathsf{DB}[1]$ otherwise. After performing h such rounds (over all pairs) for some security parameter λ, the database looks like a uniformly permuted database (when querying only $q = o(N)$ entries).

One main nice property of the Thorp shuffle is that the whole shuffle process only involves this homomorphic swap, which only depends on the database entries $\mathsf{DB}[i]$ and the random binary bits. Thus, to perform a homomorphic Thorp shuffle over a database, we first encode $\mathsf{DB}[i]$ into some valid input set $\mathcal{X}$. For example, if $\mathsf{DB}[i]$ has 3 bits (as used in the evaluation section of [23]), we can encode each entry into an element in $\mathcal{X}$ with $|\mathcal{X}| = 8$. In this case, when bootstrapping is needed (between two rounds of swapping), we only need to bootstrap an identity function over $\mathcal{X}$ instead of the entire plaintext space. This application is thus well suited to our relaxed bootstrapping (even without the closeness property).

With this high-level idea, we estimate the concrete improvement by applying our construction to such as oblivious permutation process. As shown in [23], for a database of length $N = 2^{23}$ (each entry has 3 bits), the Thorp shuffle requires 416 levels. Using our Algorithm 4, with the function being an identity function over $\mathcal{X}$ of size 8 (e.g., $\mathcal{X} = \{0, 2r, \ldots, 14r\}$, where r is the error bound), and setting the ciphertext modulus to be 860 bits (providing $\approx$ 128 bits of security), each bootstrapping of our construction allows about 13 levels of multiplications (about 400 bits of noise budget left). We can encode the entire database to $3N/32768/3 = 256$ BFV ciphertexts. Each ciphertext requires $416/13 = 32$ bootstrapping. Thus, in total, it takes $256 \times 32 \times 35 = 286720$ seconds [27], which is about 80 hours.

[27] In Table 2, we use ciphertext modulus of 650 bits instead of 860 bits. Using 860, which is essentially the maximum for 128-bit security, our bootstrapping takes about .5 seconds using GCP n4 standard.

On the other hand, in the prior works [59], encoding the entire database requires $3N/128/8 = 24576$ ciphertexts. Each bootstrapping gives ≈ 19 levels of multiplications (about 507 bits of noise budget left). Thus, in total, it takes $24576 \times 416/19 \times 22 \approx 11894784$ seconds (each of their bootstrapping takes 22 seconds), which is about 3304 hours. Similarly, it takes about 4587 hours when using [14], and about 670 hours when using [25]. Notice that [25] only offers 66-bit security, while the other two prior works and ours provide $\approx$128 bit security.

With our work, under a similar security guarantee, the bootstrapping time can be reduced by > 80x. Furthermore, since our construction supports more slots, there are less ciphertexts needed to pack the entire database. Therefore, the homomorphic Thorp shuffle can be evaluated more efficiently by having fewer regular (non-bootstrapping) ciphertext operations[28]. See [23] for a more detailed discussion and estimation.

7.2 Other Potential Applicaitons

We then discuss some other potential applications that benefit from our construction. We start with applications that do not require closeness. Such applications have a hard requirement: the input must be within $\mathcal{X}$.

PIR/PSI with computation. Private information retrieval (PIR) [19] allows one to retrieve a data entry from the database without revealing which data entry is retrieved. PIR with computation [51] further allows computing some function over the retrieved data entry (or multiple data entries for batch-PIR). Since the data entry can be encoded (e.g., as a multiple of r, the error range introduced in Section 4.1), we can use relaxed functional bootstrapping (e.g., Algorithm 1) to compute this function (or at least a sub-module of this function). Such a method can also be used for Private Set Intersection with computation [36], whose construction indeed requires a lot of bootstrapping operations, which can be replaced with our relaxed functional bootstrapping. Note that since PSI requires two-sided privacy, more careful handling is required when designing the function to not leak any database information (e.g., use noise flooding), or use other techniques like Oblivious PRF [11].

Access control. Within an organization, access control is needed to perform some action (e.g., data retrieval [10]). To realize such access control, BFV can be used in the following way. During a private data retrieval (i.e., retrieval of some documents without revealing which document it is), the user provides their identity, which corresponds to some permission level l. Then, the document also has some requirements q. The server, after obtaining q (without learning what q is) under BFV, can compute some access control function $f(q, l)$. If the result is 1, the returning ciphertext contains the document. Otherwise, the returning ciphertext contains only 0. l, q, f can all be easily encoded in a way that our relaxed functional bootstrapping supports. For example, let $|l - q| > r$ by encoding l, q accordingly, and let f checks whether $l > q$. This check can then be realized using our range function in Algorithm 5.

We then discuss some applications that may require the additional flexibility provided by closeness defined in Definition 3.2.

Secure machine learning. Machine learning (ML) using FHE has been a long-standing popular topic [44, 38, 45, 39, 20]. One major bottleneck of using FHE for ML is bootstrapping. By the nature of the fuzziness in ML, some small deviations from the model are tolerable. In this case, a relaxed functional bootstrapping satisfying 2-closeness can be used to evaluate some parts of the ML model (e.g., evaluating the activation function): any invalid inputs are simply rounded to the two nearest valid inputs. Furthermore, since the ML training process is repetitive (every batch or epoch shares the same activation function for example), our relaxed functional bootstrapping is then a perfect fit to guarantee the error budget is increased after each activation function evaluation.

Fuzzy PSI. Another natural application is fuzzy PSI [24]. Fuzzy PSI returns the elements that are similar instead of identical as in PSI. Therefore, if the fuzziness definition (e.g., the correctness definition in [62, 12]) allows the borderline elements to be decided either way, we can use our relaxed functional bootstrapping

[28]Note that a more recent version of this paper uses a larger database size. However, the analysis and our advantages remain *exactly* the same.

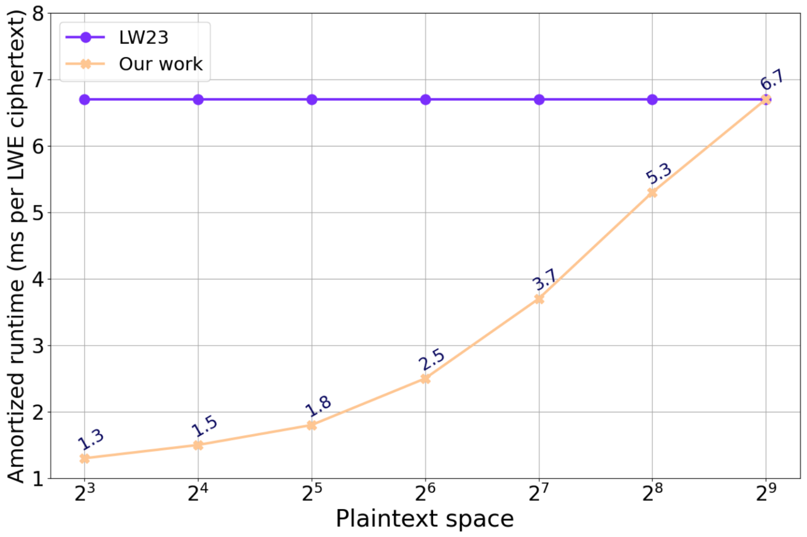

Figure 4: Comparison between our work on batched functional bootstrapping and [54].

in the following way. We first set a range $[u, v]$ to be the range for the similarity. In other words, if two elements a, b satisfying $|a - b| \in [u, v]$, the functionality should return 1; otherwise, the functionality should return 0. The issue is that the elements within $[u - r, v + r]$ may return either 0 or 1. However, due to the fuzziness definition, these are borderline cases and may be decided either way. Furthermore, with some of our constructions (Algorithm 1), the closer the difference is to $[u, v]$ the larger the probability our algorithm returns 1, which may be further preferred (discussed in Section 4.1).

There are many other potential examples, like homomorphic comparisons, private branching, fixed-point arithmetic using BFV/BGV, and so on. We leave a more detailed application-based follow-up to future works.

8 Extension to Batched FHEW/TFHE Bootstrapping

Of independent interest, our techniques can be applied to improve the batched FHEW/TFHE bootstrapping algorithm in [54]. Essentially, batched FHEW/TFHE bootstrapping is to take $2N$ LWE ciphertexts each encrypting either 0 or 1 and output the NAND result of each pair of them. Another application is that given N LWE ciphertexts each encrypting a message $x \in \mathbb{Z}_p$ for some $p \geq 2$, output an encryption of $f(x)$'s for some function f over $\mathbb{Z}_p$ that serves like a look-up table. This is called the batched functional bootstrapping.

Summary of the construction in [54]. Given $2N$ LWE ciphertexts$(\mathsf{ct}_{\mathsf{in}1,1}, \ldots, \mathsf{ct}_{\mathsf{in}1,N}), (\mathsf{ct}_{\mathsf{in}2,1}, \ldots, \mathsf{ct}_{\mathsf{in}2,N})$ [54] constructs an algorithm that outputs N LWE ciphertexts $(\mathsf{ct}_{\mathsf{out}1}, \ldots, \mathsf{ct}_{\mathsf{out}N})$ such that $\mathsf{Dec}(\mathsf{ct}_{\mathsf{out}i}) = \mathsf{NAND}(\mathsf{Dec}(\mathsf{ct}_{\mathsf{in}1,i}), \mathsf{Dec}(\mathsf{ct}_{\mathsf{in}2,i}))$ (all LWE ciphertexts encrypts $\{0, 1\}$ with plaintext space $\mathbb{Z}_3$).

The main procedure works as follows:

1. $\mathsf{ct}_i \leftarrow \mathsf{ct}_{\mathsf{in}1,i} + \mathsf{ct}_{\mathsf{in}2,i}$;
2. partial decrypt ct_i to obtain a ciphertext encrypting $m_i = \mathsf{Dec}(\mathsf{ct}_i) \in \mathbb{Z}_t$, satisfying $m_i \in (2t/3 - r/2, 2t/3 + r/2)$ (r being the error bound similar to what we have above) if $\mathsf{ct}_{1,i}$ and $\mathsf{ct}_{2,i}$ both encrypt $1 \in \mathbb{Z}_3$; otherwise $m_i \in (-r/2, r/2)$ or $\in (t/3 - r/2, t/3 + r/2)$;
3. for a NAND gate evaluation, $m_i \in (2t/3 - r/2, 2t/3 + r/2)$ is mapped to 0 and $t/3$ otherwise (which is the encoding of 1 over plaintext space $\mathbb{Z}_t$); after this step, we have a BFV ciphertext encrypting either 0 or $t/3$ in each of its N slots;

4. lastly, transform this BFV ciphertext into N LWE ciphertexts.

Optimizations. In [54], the mapping in step (3) is done via a degree-$(t-1)$ polynomial for the NAND gate. However, similar to what we have observed in our construction, this mapping has only $3r$ roots. Therefore, we replace this mapping with a degree-$(3r-1)$ polynomial, which can be evaluated much more efficiently.

For batched LWE functional bootstrapping, [54] uses a similar construction and a degree-$(t-1)$ function. Thus, for a LUT over $\log(p)$ bits, we apply our optimization and only a degree-$(\log(p) \cdot r - 1)$ function is needed.

Benchmark and comparison. For the NAND gate, since the degree now is much lower, instead of using $N = 32768$ for the underlying BFV as in [54], we use $N = 16384$ together with $\log(Q) \approx 420$, which further reduces the runtime. We use other parameters exactly the same as in [54, Section 10], except that we are using ternary keys for the LWE secret keys. The total runtime is 24.5 seconds and the amortized runtime is 1.5 ms per slot. Compared to the total runtime of 155 seconds and the amortized runtime of 4.7ms in [54][Table 2], our total runtime (i.e., the latency) is about 6x faster than [54], and the amortized runtime (i.e., the throughput) is about 3x faster.

Furthermore, we show that for p being 3-8 bits for functional bootstrapping, we achieve a speed up as shown in Fig. 4. We set $N = 32768$ for better comparison, but note that if p has only $3-5$ bits, $N = 16384$ can also be used (while still guaranteeing ~ 120 bit of security or more). It is easy to see that the runtime is roughly linear to the logarithmic size of the plaintext space. Hence, our optimization provides a better performance with smaller plaintext space.

9 Conclusion and Discussion

In this paper, we define a relaxed version of BFV bootstrapping and show how we can build a much more efficient construction under this relaxation. We also show that our technique can be applied to improve batched FHEW/TFHE ciphertexts bootstrapping. While our work shows an efficient (relaxed) BFV bootstrapping construction, there are some directions that can be further explored.

Relaxed BFV bootstrapping. One interesting direction is to build a more efficient BFV bootstrapping, e.g., by trying to explore the algebraic structure of the ciphertexts as in prior BFV works. Another interesting direction is to explore more functions that can be achieved more efficiently by relaxing some functionality of the regular bootstrapping.

Relaxed bootstrapping for other FHE schemes. One interesting question is whether can this “relaxed” version of bootstrapping help improve the efficiency of other types of FHE schemes, like CKKS. Since the main cost of CKKS bootstrapping comes from approximating the mod function using the sine function, it is not clear to us whether this relaxed correctness requirement can help. However, we believe it is still an interesting direction to investigate further.

Applications. While we have discussed many applications, we only discuss them at a high-level. It is very interesting to see an application deploy relaxed functional bootstrapping in detail. On the other hand, it is also very interesting to see more potential applications that can leverage relaxed functional bootstrapping.

Acknowledgements

We are grateful to Jaehyung Kim for answering questions regarding [41] and the anonymous reviewers for their feedback and suggestions.

References

[1] Lattigo v2.1.1. Online: `http://github.com/ldsec/lattigo`, Dec. 2020. EPFL-LDS.

[2] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[3] J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 1–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] A. A. Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. `https://eprint.iacr.org/2022/915`, commit: 122f470e0dbf94688051ab852131ccc5d26be934.

[5] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In A. Canteaut and F.-X. Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.

[6] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*, page 868–886. Springer-Verlag, 2012.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[8] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 22–25, 2011.

[9] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, Germany, Aug. 14–18, 2011.

[10] J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious transfer with access control. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM CCS 2009*, pages 131–140. ACM Press, Nov. 9–13, 2009.

[11] S. Casacuberta, J. Hesse, and A. Lehmann. SoK: Oblivious pseudorandom functions. IEEE EuroS&P 2022, 2022. `https://eprint.iacr.org/2022/302`.

[12] A. Chakraborti, M. K. Reiter, and G. C. Fanti. This paper is included in the proceedings of the 32nd usenix security symposium. In *Usenix 2023*.

[13] H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 34–54, Cham, 2019. Springer International Publishing.

[14] H. Chen and K. Han. Homomorphic lower digits removal and improved FHE bootstrapping. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 315–337. Springer, Heidelberg, Germany, Apr. 29 – May 3, 2018.

[15] J. H. Cheon, H. Choe, A. Passelègue, D. Stehlé, and E. Suvanto. Attacks against the INDCPA-d security of exact FHE schemes. CCS 2024.

[16] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.

[17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[18] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[19] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, Oct. 23–25, 1995.

[20] A. Dalvi, A. Jain, S. Moradiya, R. Nirmal, J. Sanghavi, and I. Siddavatam. Securing neural networks using homomorphic encryption. In *2021 International Conference on Intelligent Technologies (CONIT)*, pages 1–7, 2021.

[21] L. Ducas and D. Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer.

[22] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[23] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou. ThorPIR: Single server PIR via homomorphic thorp shuffles. CCS 2024, 2024.

[24] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, Germany, May 2–6, 2004.

[25] R. Geelen, I. Iliashenko, J. Kang, and F. Vercauteren. On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. Eurocrypt 2023. `https://eprint.iacr.org/2022/1364`.

[26] R. Geelen and F. Vercauteren. Bootstrapping for bgv and bfv revisited. *J. Cryptol.*, 36(2), mar 2023.

[27] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[28] C. Gentry, S. Halevi, C. Peikert, and N. P. Smart. Ring switching in bgv-style homomorphic encryption. In I. Visconti and R. De Prisco, editors, *Security and Cryptography for Networks*, pages 19–37, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[29] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, Germany, Aug. 18–22, 2013.

[30] A. Guimarães, H. V. L. Pereira, and B. van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. Asiacrypt 2023, 2023. `https://eprint.iacr.org/2023/014`.

[31] S. Halevi and V. Shoup. HElib, 2014. `https://github.com/homenc/HElib`.

[32] S. Halevi and V. Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. `https://eprint.iacr.org/2020/1481`.

[33] S. Halevi and V. Shoup. Bootstrapping for HElib. *Journal of Cryptology*, 34(1):7, Jan. 2021.

[34] K. Han, M. Hhan, and J. H. Cheon. Improved homomorphic discrete fourier transforms and fhe bootstrapping. *IEEE Access*, 7:57361–57370, 2019.

[35] K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 364–390. Springer, 2020.

[36] J. HU, J. Chen, W. Dai, and H. Wang. Fully homomorphic encryption-based protocols for enhanced private set intersection functionalities. Cryptology ePrint Archive, Paper 2023/1407, 2023. `https://eprint.iacr.org/2023/1407`.

[37] W. jie Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu. Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. SP 2021, 2020. `https://eprint.iacr.org/2020/1606`.

[38] W. jie Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 24–27, 2021.

[39] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In W. Enck and A. P. Felt, editors, *USENIX Security 2018*, pages 1651–1669. USENIX Association, Aug. 15–17, 2018.

[40] A. Kim, Y. Polyakov, and V. Zucca. Revisiting homomorphic encryption schemes for finite fields. In *ASIACRYPT 2021*, page 608–639. Springer, 2021.

[41] J. Kim, J. Seo, and Y. Song. Simpler and faster bfv bootstrapping for arbitrary plaintext modulus from ckks. Cryptology ePrint Archive, Paper 2024/109, 2024. `https://eprint.iacr.org/2024/109`.

[42] S. Kim, M. Park, J. Kim, T. Kim, and C. Min. Evalround algorithm in ckks bootstrapping. Asiacrypt 2022, 2022. `https://eprint.iacr.org/2022/1256`.

[43] D. Lee, S. Min, and Y. Song. Functional bootstrapping for packed ciphertexts via homomorphic LUT evaluation. Cryptology ePrint Archive, Paper 2024/181, 2024.

[44] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[45] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In J. Guo and R. Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 36–68, Singapore, 2023. Springer Nature Singapore.

[46] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No. *High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function*, pages 618–647. EUROCRYPT 2021, 06 2021.

[47] Y. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No. Near-optimal polynomial for modulus reduction using l2-norm for approximate homomorphic encryption. *IEEE Access*, 8:144321–144330, 2020.

[48] Y. Lee, D. Micciancio, A. Kim, R. Choi, M. Deryabin, J. Eom, and D. Yoo. Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In C. Hazay and M. Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 227–256, Cham, 2023. Springer Nature Switzerland.

[49] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. Eurocrypt 2021.

[50] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Heidelberg, Germany, Oct. 17–21, 2021.

[51] C. Lin, Z. Liu, and T. Malkin. XSPIR: Efficient symmetrically private information retrieval from ring-LWE. In V. Atluri, R. Di Pietro, C. D. Jensen, and W. Meng, editors, *ESORICS 2022, Part I*, volume 13554 of *LNCS*, pages 217–236. Springer, Heidelberg, Germany, Sept. 26–30, 2022.

[52] F.-H. Liu and H. Wang. Batch bootstrapping i: A new framework for simd bootstrapping in polynomial modulus. In C. Hazay and M. Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 321–352, Cham, 2023. Springer Nature Switzerland.

[53] F.-H. Liu and H. Wang. Batch bootstrapping i: Bootstrapping in polynomial modulus only requires o (1) fhe multiplications in amortization. In C. Hazay and M. Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 321–352, Cham, 2023. Springer Nature Switzerland.

[54] Z. Liu and Y. Wang. Amortized functional bootstrapping in less than 7ms, with $\tilde{O}(1)$ polynomial multiplications. Asiacrypt 2023. `https://eprint.iacr.org/2023/910`.

[55] Z. Liu and Y. Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. Cryptology ePrint Archive, Paper 2024/172, 2024.

[56] S. Ma, T. Huang, A. Wang, and X. Wang. Accelerating bgv bootstrapping for large p using null polynomials over $\mathbb{Z}_{p^e}$. Cryptology ePrint Archive, Paper 2024/115, 2024. `https://eprint.iacr.org/2024/115`.

[57] D. Miccianco and J. Sorrell. Ring Packing and Amortized FHEW Bootstrapping. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[58] G. D. Micheli, D. Kim, D. Micciancio, and A. Suhl. Faster amortized fhew bootstrapping using ring automorphisms. Cryptology ePrint Archive, Paper 2023/112, 2023. `https://eprint.iacr.org/2023/112`.

[59] H. Okada, R. Player, and S. Pohmann. Homomorphic polynomial evaluation using galois structure and applications to bfv bootstrapping. Asiacrypt 2023. `https://eprint.iacr.org/2023/1304`.

[60] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

[61] Microsoft SEAL, 2020. `https://github.com/Microsoft/SEAL`.

[62] E. Uzun, S. P. Chung, V. Kolesnikov, A. Boldyreva, and W. Lee. Fuzzy labeled private set intersection with applications to private real-time biometric search. In M. Bailey and R. Greenstadt, editors, *USENIX Security 2021*, pages 911–928. USENIX Association, Aug. 11–13, 2021.