
# Evaluating Larger Lookup Tables using CKKS

*or: How I learned to stop worrying about Polynomial Interpolation and love Multiplexer Tree.*

**Jules Dumezy**, Andreea Alexandru, Yuriy Polyakov, Pierre-Emmanuel Clet, Olive Chakraborty, and Aymen Boudguiga

19th March 2026

# FHE schemes

| Scheme | Plaintext | Capabilities |
|---|---|---|
| B/FV, BGV | $\mathbb{Z}_t$ | ■ Operations on $(\mathbb{Z}_t, +, \cdot)$<br>■ Amortized time/Expansion |
| DM, CGGI | $\mathbb{Z}_2, \mathbb{Z}_p$ | ■ Functional bootstrapping<br>■ Latency |
| CKKS | $\mathbb{C}$, $\mathbb{Z}_P$ | ■ Functional bootstrapping<br>■ Fixed point arithmetic<br>■ Amortized time/Expansion |

Table: Comparison of FHE schemes

# Motivation and Problem Statement

Vectorized DM/CGGI Functional bootstrapping in [BKSS24, AKP25]
Will focus comparison on [AKP25] ([BKSS24] scales similarly)
CKKS-based: degree $P$ polynomial to evaluate an arbitrary LUT on $\mathbb{Z}_P$

How does it performs?

- Amazing up to $\sim 9$ bits
- Starting at 10 bits, $2^{17}$ ring dimension
- At 16 bits: 12 hours latency and 110GB of RAM
- Same LUT for all slots

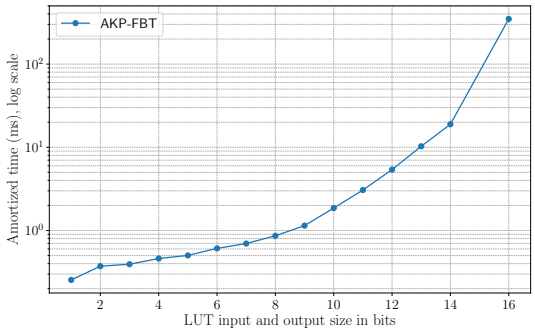



Figure: Amortized runtime of AKP-FBT

“Can we design a new functional bootstrapping algorithm to *efficiently* evaluate *large* LUTs?”

# Fast pass on discrete CKKS

Discrete CKKS: CKKS on integers $m_j = d + e$ with $d \in \mathbb{Z}_P$, $|e| \ll 1$
Polynomial interpolation to obtain $d + e'$ with $e' = o(e^2)$

Discrete CKKS functional bootstrapping

- Half BTS: StC + ModRaise + CtS
  Message + Overflow in the slots
- EvalExp: remove overflow, exponential basis

$$\exp(2i\pi(d+e)/P) \approx \zeta_P^d$$

- EvalLUT: trigonometric Hermite interpolation (degree $P$ for first order)

$$LUT(d) + e'$$

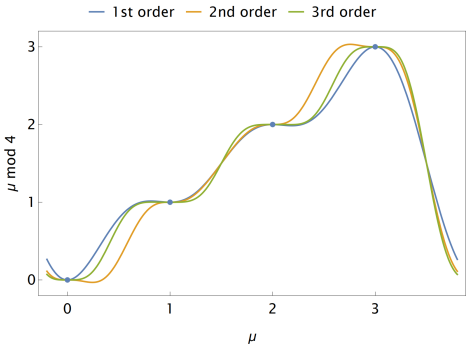



Figure: Hermite interpolation of $x \mapsto [x]_4$ (from [AKP25])

# Core idea

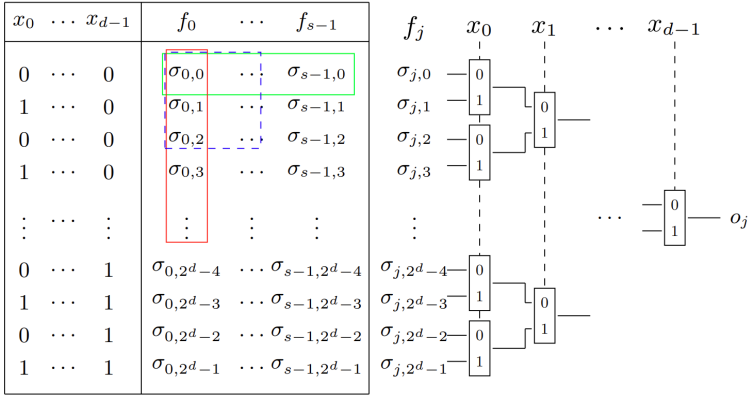



Figure: From *Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping*, Chillotti et al.

# Core idea

*Intuition*: Multiplexer tree approach adapted for CKKS

Different cost model from FHEW/TFHE: focus on minimizing multiplicative depth and ciphertext-ciphertext multiplications

Express LUT evaluation as an arithmetic circuit using binary decomposition:

$$f(x) = \sum_{i=0}^{P-1} f(i)\delta_{x,i} = \sum_{i=0}^{P-1} f(i) \prod_{j=1}^{\log P} \delta_{x_j,i_j} = \sum_{i=0}^{P-1} f(i) \prod_{j=1}^{\log P} \left(x_j \cdot i_j + (1-x_j)\cdot(1-i_j)\right)$$

Compute as a vector-matrix-vector product with Kronecker products to minimize depth and expensive operations

# Kronecker products formulation

$$f(x) = \left( \bigotimes_{i=1}^{\log \sqrt{P}} \begin{pmatrix} 1 - x_i \\ x_i \end{pmatrix} \right)^{\mathsf{T}} \mathrm{vec}^{-1}_{\sqrt{P},\sqrt{P}}\big((f(0), \ldots, f(P-1))\big) \left( \bigotimes_{i=\log \sqrt{P}+1}^{\log P} \begin{pmatrix} 1 - x_i \\ x_i \end{pmatrix} \right)$$

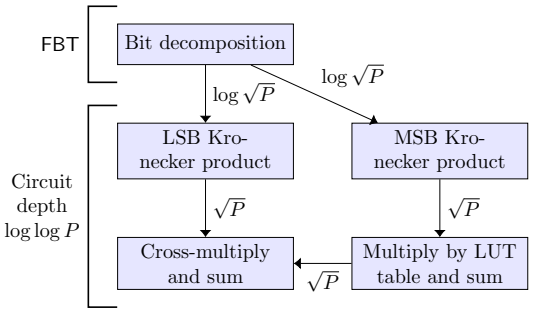
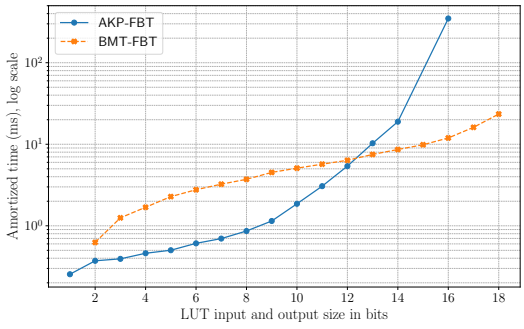
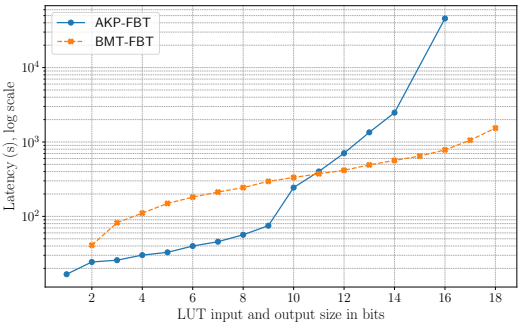



Figure: Schematic for binary multiplexer tree

# Results



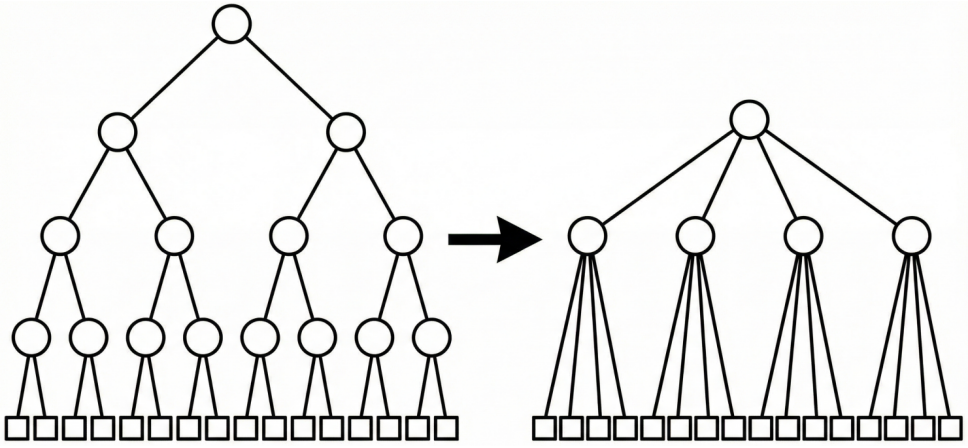

(a) Amortized time




(b) Latency

Figure: Comparison of the single-threaded amortized time (a) and latency (b) of a random LUT evaluation on a random input using AKP-FBT, BMT-FBT (less is better)

# Collapsing the multiplexer tree



# Collapsing the multiplexer tree

*Intuition*: Reduce multiplicative depth for smaller parameters and better efficiency
Choose a decomposition basis $p > 2$:

$$f(x) = \sum_{i=0}^{P-1} f(i)\delta_{x,i} = \sum_{i=0}^{P-1} f(i) \prod_{j=1}^{\log_p P} \delta_{x_j,i_j}$$

How to compute the $\delta_{x_j,i_j}$ ? The functions $x \mapsto \delta_{x,i}$ for $i \in \mathbb{Z}_p$ can be seen as $p$-to-$p$ LUT

*Functional digit decomposition* to evaluate (multiple) arbitrary LUTs on each extracted digit
$\rightarrow$ Outputs $(x_j, \delta_{x_j,0}, \delta_{x_j,1}, \ldots, \delta_{x_j,p-1})$

# Kronecker products formulation

$$f(x) = \left( \bigotimes_{i=1}^{\log_p \sqrt{P}} \begin{pmatrix} \delta_{x_i,0} \\ \vdots \\ \delta_{x_i,p-1} \end{pmatrix} \right)^{\mathsf{T}} \mathrm{vec}^{-1}_{\sqrt{P},\sqrt{P}}\big((f(0),\ldots,f(P-1))\big) \left( \bigotimes_{i=\log_p \sqrt{P}+1}^{\log_p P} \begin{pmatrix} \delta_{x_i,0} \\ \vdots \\ \delta_{x_i,p-1} \end{pmatrix} \right)$$

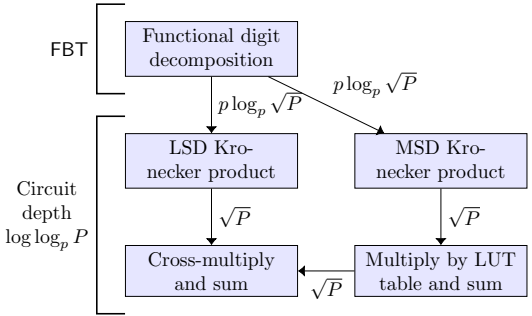
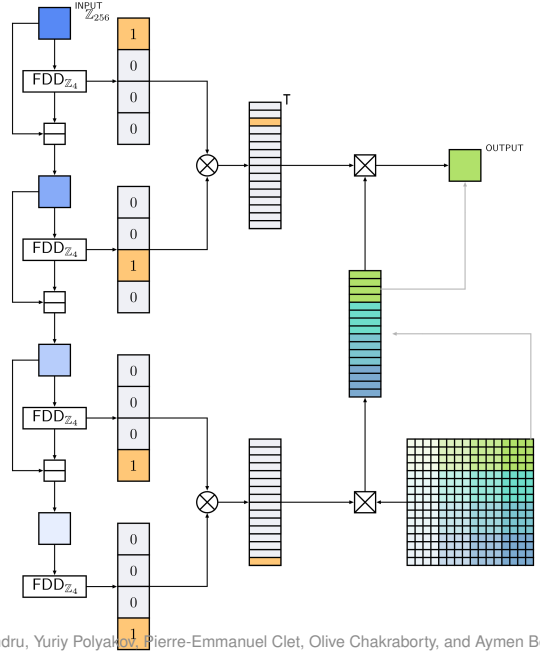
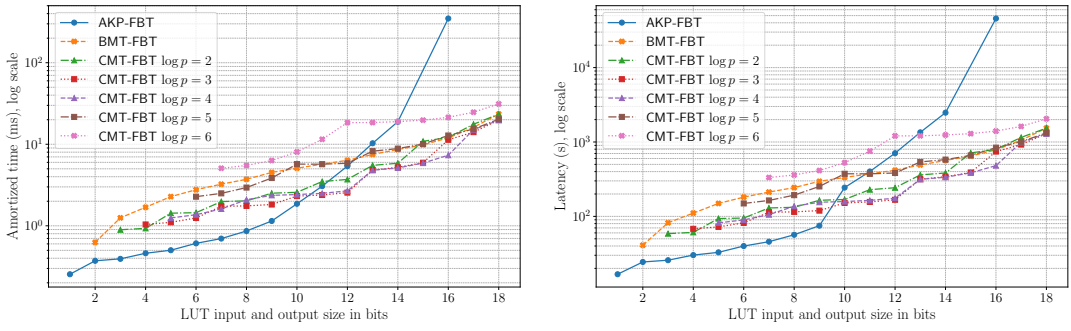



Figure: Schematic for collapsed multiplexer tree

# Example


INPUT
$\mathbb{Z}_{256}$
$\mathrm{FDD}_{\mathbb{Z}_4}$
$\mathrm{FDD}_{\mathbb{Z}_4}$
$\mathrm{FDD}_{\mathbb{Z}_4}$
$\mathrm{FDD}_{\mathbb{Z}_4}$
T
OUTPUT

# Results

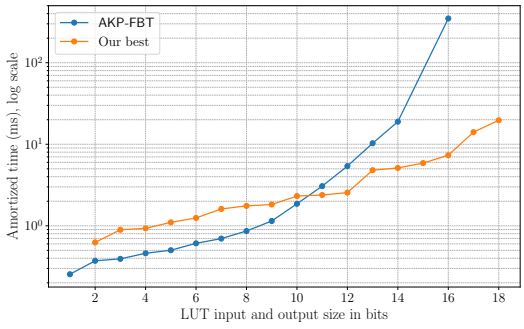



(a) Amortized time

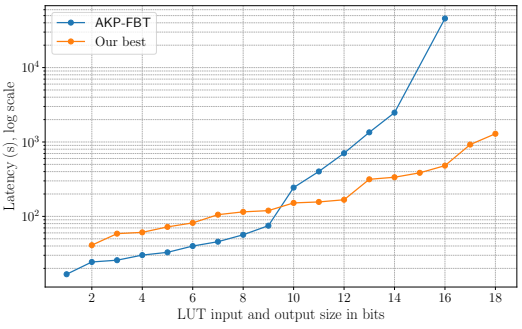
(b) Latency

Figure: Comparison of the single-threaded amortized time (a) and latency (b) of a random LUT evaluation on a random input using AKP-FBT, BMT-FBT and CMT-FBT for different digit size

# Results




(a) Amortized time




(b) Latency

Figure: Comparison of the single-threaded amortized time (a) and latency (b) of a random LUT evaluation on a random input using AKP-FBT, BMT-FBT and CMT-FBT for different digit size

# Additional results

| Method | Ring dimension | Amtz. time (ms) | Latency |
|---|---|---|---|
| AKP-FBT | $2^{17}$ | 349 | 12h40 |
| BMT-FBT | $2^{16}$ | 11.9 | 13 min |
| CMT-FBT | $2^{16}$ | 7.34 | 8 min |
| CMT-FBT (MT) | $2^{16}$ | 1.71 | < 2 min |

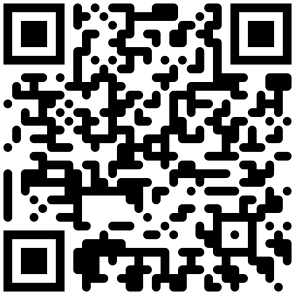
Table: Comparison of functional bootstrapping algorithms for 16-bit LUTs

Notable extensions:

- Different LUTs for different slots of the ciphertext
- Encrypted LUTs
- CMT-FBT with itself as a subroutine to target even larger LUTs (32-bit)

# Conclusion

- Large LUT evaluation will be added in an upcoming release of OpenFHE
- Pushed in-house to LUT sizes of 22/24 bits $\rightarrow$ the main bottleneck is RAM
- Optimizations and exact algorithmic details are given in the paper available on eprint 2025/1301
- See you at CHES'26 in October!



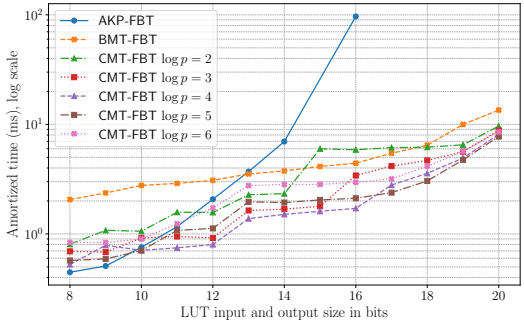

# Thank you !

**CEA Nano-INNOV**
91 120 Palaiseau Cedex
France
jules.dumezy@cea.fr

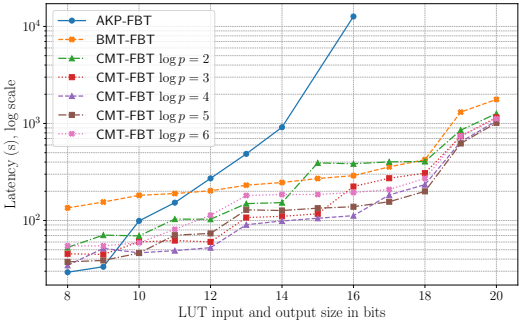
Example of Kronecker delta computation for a $16$-to-$16$ LUT in base $2$:

$$
\begin{aligned}
f(11) = f(2^0 + 2^1 + 2^3) &= \left[ \overset{2^3}{\begin{pmatrix} 0 \\ 1 \end{pmatrix}} \otimes \overset{2^2}{\begin{pmatrix} 1 \\ 0 \end{pmatrix}} \right]^\intercal \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \left[ \overset{2^1}{\begin{pmatrix} 0 \\ 1 \end{pmatrix}} \otimes \overset{2^0}{\begin{pmatrix} 0 \\ 1 \end{pmatrix}} \right] \\
&= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 7 \\ 11 \\ 15 \end{pmatrix} \\
&= 11
\end{aligned}
$$

# Results with parallelization



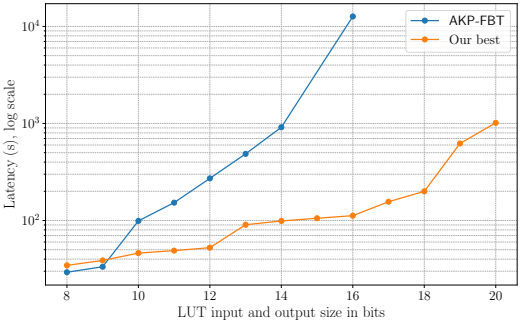

(a) Amortized time



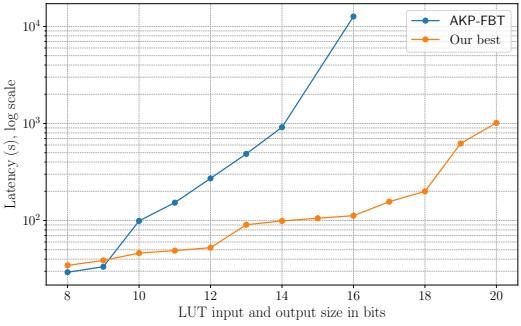

(b) Latency

Figure: Comparison of the multi-threaded amortized time (a) and latency (b) of a random LUT evaluation on a random input using AKP-FBT, BMT-FBT and CMT-FBT for different digit size

# Results with parallelization




(a) Amortized time




(b) Latency

Figure: Comparison of the multi-threaded amortized time (a) and latency (b) of a random LUT evaluation on a random input using AKP-FBT, BMT-FBT and CMT-FBT for different digit size

# Memory Consumption

Table: Comparison of peak RAM consumption for the evaluation of LUTs on $\mathbb{Z}_P$ in the single-threaded setting. We highlight the smallest memory consumption in bold.

| $\log P$ | 10 | 12 | 14 | 16 |
|---|---|---|---|---|
| AKP-FBT Peak RAM (GB) | 22.9 | 27.4 | 32.3 | 110.2 |
| BMT-FBT Peak RAM (GB) | 22.4 | 22.6 | **24.8** | **26.2** |
| CMT-FBT Peak RAM (GB) | **13.8** | **13.9** | 27.3 | 28.5 |