Breaking the Size Barrier: Universal Circuits meet Lookup Tables

Yann Disser, Daniel Günther, Thomas Schneider, Maximilian Stillger, Arthur Wigandt, Hossein Yalame

Technical University of Darmstadt, Darmstadt, Germany
disser@mathematik.tu-darmstadt.de, {guenther,schneider, yalame}@encrypto.cs.tu-darmstadt.de, maximilian.stillger@arcor.de, arthur.wigandt@protonmail.com

Abstract. A Universal Circuit (UC) is a Boolean circuit of size $\Theta(n \log n)$ that can simulate any Boolean function up to a certain size n. Valiant (STOC'76) provided the first two UC constructions of asymptotic sizes $\sim 5n \log n$ and $\sim 4.75n \log n$, and today's most efficient construction of Liu et al. (CRYPTO'21) has size $\sim 3n \log n$. Evaluating a public UC with a secure Multi-Party Computation (MPC) protocol allows efficient Private Function Evaluation (PFE), where a private function is evaluated on private data.
Previously, most UC constructions have only been developed for circuits consisting of 2-input gates. In this work, we expand upon this by considering UCs that simulate circuits made up of $(\rho \to \omega)$-Lookup Tables (LUTs) that map ρ input bits to ω output bits. Our LUT-based UC (LUC) construction has an asymptotic size of $1.5\rho\omega n \log \omega n$ and improves the size of the UC over the best previous UC construction of Liu et al. (CRYPTO'21) by factors $1.13\times$ - $2.18\times$ for common functions. Our results show that the greatest size improvement is achieved for $\rho = 3$, and it decreases for $\rho > 4$.
Furthermore, we introduce Varying Universal Circuits (VUCs), which reduce circuit size at the expense of leaking the number of inputs ρ and outputs ω of each LUT. Our benchmarks demonstrate that VUCs can improve over the size of the LUC construction by a factor of up to $1.45\times$.

Keywords: universal circuit, private function evaluation, multi-party computation

1 Introduction

A *Universal Circuit* (UC) $\mathcal{U}$ is a Boolean circuit that can simulate any Boolean circuit C consisting of n_i inputs, n_g gates, and n_o outputs. The UC $\mathcal{U}$ takes, in addition to the function's input x, a set of programming bits p^C defining the circuit C that $\mathcal{U}$ simulates, i.e., the UC computes $\mathcal{U}(x, p^C) = C(x)$.

The first two UC constructions known as *2-way* and *4-way* split UCs with asymptotic optimal size of $\Theta(n \log n)$ and depth of $\mathcal{O}(n)$ were proposed by Valiant [44], where $n = n_i + n_g + n_o$ refers to the size of the simulated circuit C.

Cook and Hover [9] designed a depth-optimized UC construction for simulating Boolean circuits of size n and depth d that has size $\mathcal{O}(n^3 d/n)$ and depth $\mathcal{O}(d)$. The first practical implementation of a UC of non-optimal asymptotic size $\mathcal{O}(n \log^2 n)$ was given by Kolesnikov and Schneider [29]. A line of work [27,31,19,3,50] followed with the common goal to minimize the size of Valiant's UC construction. Recently, Liu et al. [32] provided a UC construction with today's most efficient concrete size of $\sim 3n \log n$.

All of these works designed UCs to simulate Boolean gates having at most 2 inputs and 1 output. However, Valiant's UC construction can easily be extended to simulate circuits consisting of $(\rho \to 1)$-LUT, namely *Lookup-Tables* that have ρ inputs $x_1, \ldots, x_\rho$ and one output y, and can be programmed to compute $y = f(x_1, \ldots, x_\rho)$ for an arbitrary function f [41]. Our goal is to propose LUT-based UCs (LUC) to support $(\rho \to \omega)$-LUTs, which have in total ω output bits $y_1, \ldots, y_\omega$ and are programmed to compute $y_i = f^i(x_1, \ldots, x_\rho)$ for $1 \leq i \leq \rho$ and an arbitrary function f^i. In addition, we introduce *Varying UCs (VUCs)*, a departure from conventional UCs that can simulate circuits concisting of $(\rho \to \omega)$-LUTs with varying numbers of inputs ρ and outputs ω, thereby leaking the arity of each LUT. VUCs have various applications (summarized in §1.1) like logic locking [48], which enables the designer to provide the foundry with a "locked" version of the original circuit. Once the locked circuit is fabricated, authorized users can regain access to the original functionality by using a secret key.

On top of our new UC constructions, we provide implementations of our constructions[1] and analyze the size optimization of simulating LUT-based circuits with LUCs and VUCs compared to simulating equivalent Boolean circuits.

1.1 Applications of (Varying) Universal Circuits

The most prominent application for UCs is *Private Function Evaluation* (PFE) [2], which can be seen as a generalization of *Secure Multi-Party Computation* (MPC) [47,17]. In MPC, a set of k parties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ jointly compute a publicly known circuit C on their respective private inputs $x_1, \ldots, x_k$ and obtain nothing but the result $C(x_1, \ldots, x_k)$. In PFE, the circuit C that shall be computed is private information as well, i.e., party $\mathcal{P}_1$ with circuit input C and parties $\mathcal{P}_{2 \leq i \leq k}$ with data inputs $x_2, \ldots, x_k$ run a protocol that yields nothing but $C(x_2, \ldots, x_k)$ and parties $\mathcal{P}_{2 \leq i \leq k}$ does not learn any information about the circuit C.

PFE can be implemented via MPC by means of UCs as follows: The parties $\mathcal{P}_1, \ldots, \mathcal{P}_k$ run an MPC protocol that evaluates the universal circuit $\mathcal{U}$ as public circuit on the secret inputs p^C of party $\mathcal{P}_1$ and $x_2, \ldots, x_k$ of parties $\mathcal{P}_2, \ldots, \mathcal{P}_k$, resulting in $\mathcal{U}(p^C, x_2, ..., x_k) = C(x_2, ..., x_k)$. In summary, PFE based on UCs is a very generic approach. It can simply be plugged into arbitrary MPC frameworks without any modification to the underlying MPC protocol, resulting in the same security level (semi-honest, covert, or malicious) as the underlying MPC framework. In addition, PFE is completely compatible with the features included in MPC like

[1] Our implementation is open-sourced under the MIT License and can be reviewed anonymously here: https://anonymous.4open.science/r/CRYPTO-LUC.

secure outsourcing [23] and *non-interactive computation* [31]. PFE is applicable for situations where customers aim to use a service from companies who want to hide *how* they perform the computation and do not learn the customer's data.[2] These include privacy-preserving credit checking for credit worthiness [14], software diagnosis [8], medical diagnosis [5], and insurance tariffs [18] to name a few.

As a trade-off between privacy and efficiency, a weaker variant of PFE called Semi-Private Function Evaluation (SPFE) [18,37] has been developed. Unlike PFE, SPFE doesn't conceal the entire function but leaks the topology of certain sub-functions. SPFE can be applied in PFE use cases that allow the exposure of certain function components, such as situations where some function information is already leaked, for advertising purposes like, e.g., car insurance companies give discounts for elder people. Beyond (S)PFE, UCs have many other applications like hiding the policy circuit in attribute-based encryption [4,15], multi-hop homomorphic encryption [16], verifiable computation [13], program obfuscation [51], and recently hardware logic locking [34].

Virtually Private Function Evaluation (VPFE). In this work, we introduce *Virtually Private Function Evaluation* (VPFE) whose privacy-guarantee is a trade-off between PFE and SPFE. Concretely, like in PFE, our Varying UCs (VUCs) for VPFE hide the topology and the gate functionality of the circuit, but leak the number of in- and outputs of the simulated LUTs. This has applications in logic obfuscation techniques called logic locking, as demonstrated in prior studies such as LUT-Lock [22] and eFPGA [7]. These techniques proposed using LUTs to achieve secure logic locking on Application-Specific Integrated Circuit (ASIC) designs by removing critical elements and mapping them to custom LUTs. As shown in [7, Fig. 3], the adversary can only determine the number of inputs and outputs of a LUT, while the LUT's configuration bits are hidden, which is exactly our setting for VPFE using VUCs. Without this knowledge, there is no adversarial information leakage [7, Tab. 5]. Therefore, our VUCs can be used for secure logic locking.

1.2 Related Work

Universal Circuits (UCs). Valiant [44] defined universal circuits, showed that they have a lower bound of size $\Omega(n \log n)$, and proposed two asymptotically size-optimal constructions using a 2-way or a 4-way recursive structure of sizes $\sim 5n \log n$ and $\sim 4.75n \log n$, respectively. Hence, relevant research challenges left are reducing the prefactor and the concrete UC sizes. Valiant's constructions can easily be extended to simulate $(\rho \to 1)$-LUT circuit simulations (cf. construction in §4) as shown by Sadeghi and Schneider [41, App. A].

A modular UC construction of non-optimal size $1.5n \log^2 n + 2.5n \log n$ proposed and implemented in Kolensikov and Schneider [29]. Their construction beats Valiant's construction for small circuits thanks to small prefactors. Motivated to provide more efficient PFE, Kiss and Schneider [27] implemented Valiant's 2-way

[2] In contrast to Fully Homomorphic Encryption (FHE) which can also be used for PFE, UC-based PFE can be based on mostly symmetric encryption and requires concretely much less computation.

split construction and proposed a more efficient hybrid construction combining the 2-way split construction with the modular construction of [29]. In a concurrent work, Lipmaa et al. [31] generalized Valiant's construction to a k-way split construction and proved that the optimal value for k is 3.147, i.e., $k \in \{3, 4\}$ when k is an integer. Günther et al. [19] modularized Valiant's construction, implemented the more efficient 4-way split construction, gave a generic edge-embedding algorithm for general k-way split constructions, and showed that the 3-way split construction with Valiant's framework is less efficient than the 2-way split construction. Zhao et al. [50] improved Valiant's 4-way split construction to size $\sim 4.5n \log n$, which is today's most efficient asymptotic size for UCs in Valiant's framework. Alhassan et al. [3] proposed and implemented a scalable hybrid UC construction combining Valiant's 2-way and the 4-way construction with Zhao et al.'s improvements [50]. Most recently, Liu et al. [32] reduced redundancies in Valiant's framework and provided today's most efficient UC construction of size $\sim 3n \log n$ based on Valiant's 2-way construction, showed that $k =$ 2-way split is the most efficient in their new UC framework, and already almost reached their computed lower bound of $\sim 2.95n \log n$. We provide the first implementation of their construction and use it as a basis for our UC constructions for LUT-based circuits.

Private Function Evaluation (PFE). Katz and Malka [26] designed a constant-round two-party PFE protocol with linear communication complexity based on homomorphic public-key encryption. Their protocol was optimized and implemented by Holz et al. [21] who showed that it outperforms the hybrid UC implementation of Alhassan et al. [3] for circuit sizes with $n = 1$ million gates. However, their protocol is not generic and hence not directly compatible with arbitrary MPC frameworks, which makes it less flexible. For instance, their protocol cannot easily be extended to multiple parties. In fact, recent PFE applications relied on so-called Semi-Private Function Evaluation (SPFE) where not necessarily the whole function needs to be hidden from the other parties, but selected parts of the function can be leaked. The first SPFE construction was proposed by Paus et al. [37] who provided some general building blocks that can be programmed with one function out of a class of functions (e.g., ADD/SUB whose circuits have the same topology). Recently, Günther et al. [18] developed an SPFE framework that allows to split the function into public and private components, embed the private components into UCs, and merge them into one Boolean circuit that is evaluated via MPC. They demonstrated their framework on computing car insurance tariffs and observed that some information of the function is public, e.g., that older people usually get discounts due to their experience.

MPC on LUTs. In the area of secure multi-party computation, prior work noticed that 2-input/1-output gates can be extended into multi-input/multi-output gates to reduce the circuit evaluation overhead [33,20,39,11,35]. In Yao's GC setting, Fairplay [33] implemented MPC protocols to evaluate gates with up to 3-input gates. The TASTY framework [20] implemented ρ-input garbled gates using the garbled row reduction optimization [38]. Recently, [39] proposed an MPC protocol that works on circuits with multi-input/multi-output gates instead of working on circuits with 2-input gates. Another line of work is the secret-sharing setting, where the motivation is to optimize the rounds and communication of the online phase without the use of Yao's GC protocol. [11] extended 2-input AND gates to the general N-input case using lookup tables (LUTs). Recently, ABY2.0 [35] extended AND gates from the 2-input to the multi-input setting with a constant online communication complexity at the cost of exponential offline communication in the number of inputs. In addition, Syncirc [36] handles the circuit generation with multi-input gates by using industry-grade hardware synthesis tools [46,6].

1.3 Outline and Our Contributions

So far, UC-based PFE research considered synthesis of the input circuit (to generate a smaller number of 2-input gates) and construction of the UC (to minimize its size) as independent tasks. In our work, we show that using multi-input/output LUTs these two tasks can be combined to yield a better size. After giving the preliminaries for universal circuits in §2 and summarizing the two UC constructions of Valiant [44] (§3.2) and Liu et al. [32] (§3.3), we contribute the following:
LUT-based UC (LUC) Construction (S§ 4). Valiant's UC construction can easily be extended to support the evaluation of $(\rho \rightarrow 1)$-LUT-based circuits by merging for the ρ inputs ρ instances of its basic building block called edge-universal graph [41, App. A]. This leads to a total size of $\sim 1.5\rho n \log n$ using Liu et al.'s [32] UC construction. In this work, we provide a novel technique to extend this construction to simulate $(\rho \rightarrow \omega)$-LUT-based functions with ρ inputs and ω outputs. This technique is general, can be applied to all UC constructions based on Valiant's framework [44] and improvements by Liu et al. [32], and fits into the definition of UCs (cf. Def. 1).
Size Improvements of UCs for LUT-based Basic Primitives (§4.3). Taking PFE as our greatest motivation for improving UC sizes, we study three basic building blocks which can be used to construct more complex functionalities for common PFE applications: We compare the asymptotic circuit sizes when evaluating our new LUC construction with UCs for equivalent binary gates (cf. Tab. 2) and achieve size improvements of factor $2\times$ for full adders, $2.67\times$ for comparisons, and $2\times$ for multiplexers. Tab. 1 shows the history of improvements in UC sizes.
Varying UC (VUC) Construction (§5). There are applications (cf., §1.1) where only the programming of the LUTs need to be hidden, but not their arities, i.e., we can leak the LUT types in our circuit. For this we introduce *Varying Universal Circuits* (VUC) which are circuits that can simulate other LUT-based circuits while hiding their topology and the LUT programmings, but leak the sequence of arities of the LUTs. We give the first VUC construction that eliminates the leading ρ factor of our LUC construction, while still maintaining its general design, i.e., we can transform all UC constructions to our new VUC construction.
Implementation (§6.1. We provide the first implementation of today's most efficient UC construction of Liu et al. [32] which is of independent interest[3]. Moreover, we extend it with multi-input and multi-output LUT support and

[3] It can be reviewed anonymously at: `https://anonymous.4open.science/r/CRYPTO-LUC` and we will publish it as open source upon acceptance of our paper.

integrate it into the MPC framework ABY [10] for PFE. To create LUT-based circuits, we used the hardware circuit synthesis tool Yosys-ABC [46,6] and Design Compiler [1] for LUT-Mapping. Our LUT-based PFE is significantly optimized by combining LUTs with overlapping inputs and multiple outputs. However, hardware synthesis tools do not by default support mapping to multiple output LUTs. In order to address this, we post-process the single-output LUT circuits produced by the synthesis tool in order to convert them to multi-output LUT circuits.
Evaluation (§6.2). We experimentally evaluate our LUC and VUC constructions for various LUT sizes, and compare them with the the previous best construction of Liu et al. [32]. The asymptotic UC sizes and improvements over previous works are given in Tab. 4 for LUC and in Tab. 6 for VUC. Our new LUC constructions outperform the state-of-the-art UC-based PFE of [32] in terms of circuit sizes up to 2.18×.

Table 1: Asymptotic sizes of various UC constructions and improvements over previous works. Previous UCs were for 2-input gates and LUC is for ρ-input LUTs. So, the size of LUC is function-dependent and its improvement refers to representing an equivalent function consisting of only 2-input gates.

Universal Circuit	Asymptotic Size	Improvement over best previous work
Valiant's 2-way [44]	$5n\log n$	-
Valiant's 4-way [44]	$4.75n\log n$	$1.05\times$
Zhao et al.'s 4-way [50]	$4.5n\log n$	$1.06\times$
Liu et al.'s 2-way [32]	$3n\log n$	$1.5\times$
Our LUC	$\approx \mathbf{1.5}n\log n$	$\mathbf{1.13-2.18}\times$

2 Preliminaries

We refer to the size of a circuit n as the sum of its number of inputs n_i, gates n_g, and outputs n_o.

Definition 1 (Universal Circuit [44,3]). *A Universal Circuit $\mathcal{U}$ for n_i inputs, n_g gates, and n_o outputs is a Boolean circuit that can be programmed to compute any Boolean circuit C with n_i inputs, n_g gates and n_o outputs by defining a set of programming bits p^C such that $\mathcal{U}(x, p^C) = C(x)$ for all possible input values $x \in \{0,1\}^{n_i}$.*

2.1 Graph Theory

Let $G = (V, E)$ be a directed graph and $v \in V$. The indegree (resp. outdegree) of v, the number of incoming (resp. outgoing) edges, is denoted by $\deg^+(v)$ (resp. $\deg^-(v)$). G has fanin (resp. fanout) ρ if $\deg^+(v) \le \rho$ (resp. $\deg^-(v) \le \rho$) for all $v \in V$. We denote by $\Gamma_\rho(n)$ all directed acyclic graphs with at most n nodes and

fanin/fanout ρ for $\rho, n \in \mathbb{N}$. For $U \subset V$, $G[U] \coloneqq \{U, \{e = (u, v) \in E : u, v \in U\}\}$ denotes the subgraph induced by U. We omit the index G in the above definitions if G is clear from the context.
Let $G = (V, E) \in \Gamma_\rho(n)$. A topological order for G is a map $\eta_G \colon V \to \{1, ..., |V|\}$ such that $\forall\ (u, v) \in E : \eta_G(u) < \eta_G(v)$.
We represent Boolean circuits as directed acyclic graph $G \in \Gamma_\rho(n)$ for some $\rho > 1$. However, almost all previous works [44,27,31,19,50,32] restricted the circuits, that are simulated via UCs, to fanin/fanout $\rho = 2$. The reason for this restriction can be found in the structure of universal circuits according to Valiant's [44] and Liu et al.'s [32] frameworks. On a high level, a universal circuit (UC) for simulating circuits $C \in \Gamma_\rho(n)$ is composed of ρ so-called *Edge-Universal Graphs* (EUGs) each of size $\mathcal{O}(n \log n)$, i.e., the total size of the UC grows linearly with the maximum fanin/fanout ρ of the gates in the simulated circuit C.

Definition 2 (Edge-Embedding [44,31,3,32]). *Let $G = (V, E, P)$ and $G' = (P, E')$ be directed graphs with $P \subset V$ and G' acyclic. An edge-embedding from G' into G is a map $\psi \colon E' \to \mathcal{P}_G$, where $\mathcal{P}_G$ denotes the set of all paths in G, with the following properties:*

- *$\psi(e')$ is a u-v-path (in G) for all $e' = (u, v) \in E'$,*
- *$\psi(e')$ and $\psi(\tilde{e}')$ are edge-disjoint paths for all $e', \tilde{e}' \in E'$ with $e' \neq \tilde{e}'$.*

Definition 3 (Edge-Universal Graph [44,31,3,32]). *A directed graph $G = (V, E, P)$, denoted as $\mathcal{U}_\rho(n)$ with ordered pole set $P \coloneqq \{p_1, ..., p_n\} \subset V$ is called an Edge-Universal Graph for $\Gamma_\rho(n)$ if:*

- *G is acyclic.*
- *Every acyclic $G' = (P, E') \in \Gamma_\rho(n)$ that is order-preserving, i.e., $\forall\ e = (p_i, p_j) \in E' \Rightarrow i < j$, can be edge-embedded into G.*

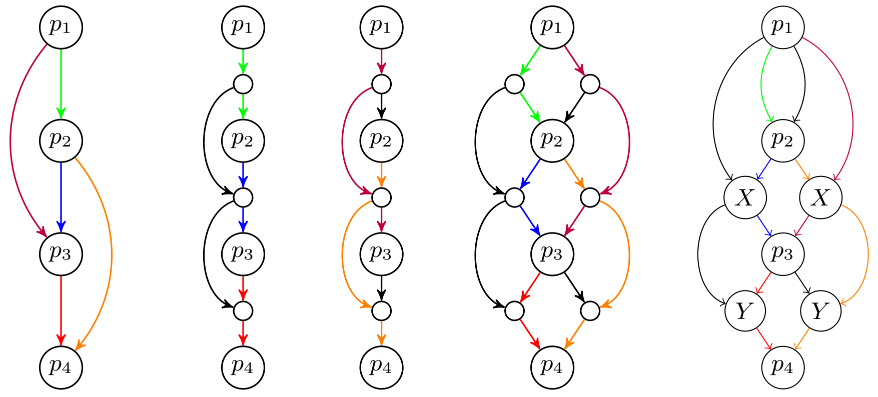
On a high level, the graph $G' = (P, E')$ in Defs. 2 and 3 represents a Boolean function that is embedded into the graph $G = (V, E, P)$, which represents the UC, where $P \subset V$ is the pole set of size $|P| = n$, which represents the inputs, gates and outputs of the function represented in G'. As an EUG requires that every $G' \in \Gamma_\rho(n)$ can be edge-embedded into G, the UC built by the EUG can compute any function represented by a graph in the set $\Gamma_\rho(n)$.

EUGs for $\Gamma_2(n)$ graphs were constructed by merging two EUGs for $\Gamma_1(n)$ graphs (cf. Def. 4 and Fig. 1)[44,27,31,19,50,3,32]. Thus, research focused on minimizing the size of general EUGs for $\Gamma_1(n)$ graphs as these can be merged to EUGs for arbitrary $\Gamma_\rho(n)$ graphs by merging ρ instances of $\Gamma_1(n)$ EUGs (cf. Cor. 1).

Definition 4 (Merging of EUG). *Let $G = (V, E, P)$ and $\bar{G} = (\bar{V}, \bar{E}, P)$ be two EUG for $\Gamma_\rho(n)$ and $\Gamma_{\bar{\rho}}(n)$ with the same pole order and $V \cap \bar{V} = P$. Then $\hat{G} = (V \cup \bar{V}, E \cup \bar{E}, P)$ is called the merging of G and $\bar{G}$.*

Proposition 1. *The merging of a $\Gamma_\rho(n)$ and a $\Gamma_{\bar{\rho}}(n)$ EUG is a $\Gamma_{\rho+\bar{\rho}}(n)$ EUG.*

We proof Proposition 1 in App. A.



(a) $\Gamma_2(4)$ graph (b) $\mathcal{U}_1(4)$ (c) $\mathcal{U}_1(4)$ (d) merged $\mathcal{U}_2(4)$ (e) UC

Fig. 1: (a) shows the $\Gamma_2(4)$ graph with already partitioned edge sets E_1 and E_2, (b) and (c) show the EUGs in which the edge sets E_1 resp. E_2 are embedded, (d) shows the merged EUG with all edges embedded, (e) shows the resulting UC, where p_1 is an input, and p_2, p_3, p_4 are translated to universal gates.

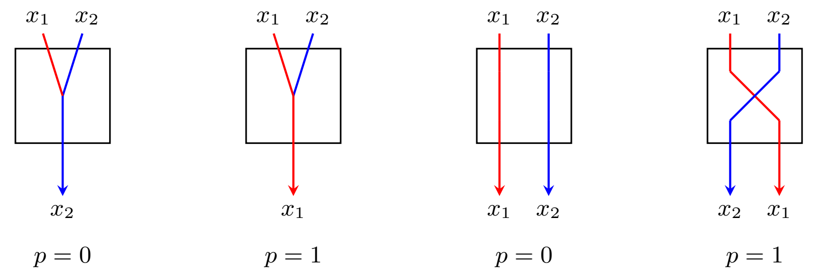
Corollary 1 ([44, Corollary 2.2]). *An EUG for $\Gamma_\rho(n)$ can be constructed by merging ρ EUGs for $\Gamma_1(n)$.*

Proof. Let $G = (V, E, P)$ be a $\Gamma_1(n)$ EUG. Create $\rho - 1$ copies of G with the same pole set and merge these graphs successively. Correctness follows directly by applying Prop. 1 ρ times. □

We call the UCs that are constructed according to Cor. 1 *LUT-based UCs* (LUCs) and this construction was first mentioned in [41, App. A]. However, in §5, we introduce our so-called *Varying UC (VUC) construction* that is constructed by two instances of $\Gamma_1(n)$ EUGs but still allows to edge-embed graphs with arbitrary fanin ρ.

2.2 Building Universal Circuits from Edge-Universal Graphs

Boolean Circuits. A Boolean circuit can be seen as a directed acyclic graph whose nodes are Boolean inputs, (binary) gates, and outputs, and the directed edges are the wires. A Boolean gate is a function $z\colon \{0,1\}^k \to \{0,1\}$ for $k \in \mathbb{N}$. However, we can always divide a k-input gate into $\mathcal{O}(2^k)$ binary gates using Shannon's expansion theorem [42]. Unfortunately, we cannot avoid an exponential blow-up of the number of gates by this transformation [45, Theorem 2.1]. The two most prominent minimization methods for Boolean circuits are due to Karnaugh [25] and Quine-McCluskey [40]. As already mentioned, the UC constructions by Valiant [44] and Liu et al. [32] are designed to embed $\Gamma_\rho(n)$ graphs, thus we possibly need to reduce the outdegree of the gates to ρ by using



(a) Y-switch (b) X-switch

Fig. 2: Switching nodes with programming bit p.

so-called copy gates which just copy their inputs [44, Corollary 3.1].[4]

From Edge-Universal Graphs to Universal Circuits. The translation from a EUG $G = (V, E, P)$ into a UC is depicted in Fig. 1 and works as follows. First, the nodes of circuit C that shall be embedded into G constitute the set of poles P of the EUG. A pole $p \in P$ is translated into an input (resp. output) wire, if p corresponds to an input (resp. output) in C, or into a so-called *Universal Gate*, if p corresponds to a gate in C. Universal gates take k inputs ($k = 2$ in the previous works [44,3,32]), 2^k programming bits, compute one output and can be programmed to simulate any k input Boolean gate by specifying the truth table with the programming bits. We can implement universal gates with a binary tree of $2^k - 1$ multiplexers (also called Y-switches) spanned over the 2^k programming bits, where the correct programming bit specified by the k inputs is forwarded to the output (we refer to [44,27,19,50,3,32] for more details).[5]

The remaining nodes in the set $V \setminus P$ are for connecting the routes between the poles. A node $v \in V \setminus P$ is translated as follows:

– if v has two incoming edges and one outgoing edge, it is translated into a multiplexer/Y-switch (cf. Fig. 2a). A multiplexer has two inputs x_0 and x_1 and a programming bit p and outputs one bit, namely x_p. It is implemented with 1 AND gate and 2 XOR gates [29].

– if v has two incoming edges and two outgoing edges, it is translated into an X-switch (cf. Fig. 2b). An X-switch has two inputs x_0 and x_1, one programming bit p and outputs two bits, namely (x_p, x_{1-p}). It is implemented with 1 AND gate and 3 XOR gates [29].

– if v has one incoming wire, it is replaced by a single wire that connects all of the outgoing edges.

The programming bits of the nodes are derived from the edge-embedding.

[4] Note that a Universal Circuit can also compute circuits with less than the specified number of inputs, gates, and outputs by using dummy values with no functionality.

[5] When evaluating the UC with Yao's garbled circuit protocol [47], the universal gate can be directly implemented as a garbled table when the function holder takes over the garbling part.

3 UC Constructions

In this section, we summarize the general guidelines for constructing edge-universal graphs (§ 3.1), present the original idea of Valiant [44] (§ 3.2), and describe the state-of-the-art construction of Liu et al. [32] (§ 3.3).

3.1 General EUG constructions

The strategy for building UCs via EUGs is to construct $\Gamma_1(n)$ EUGs of smallest size, merging ρ instances of these (cf. Cor. 1) to construct a $\Gamma_\rho(n)$ EUG ($\rho = 2$ for binary gates/circuits), and translating this into a universal circuit. Valiant [44] proposed the first two constructions for $\Gamma_1(n)$ EUGs, today known as 2-way and 4-way constructions, having asymptotic sizes of $\sim 2.5n \log n$ and $\sim 2.375n \log n$.[6] Recently, Liu et al. [32] extended Valiant's framework, simplified the construction, and achieved an EUG based on the 2-way approach of asymptotically optimal size of $\sim 1.5n \log n$, which almost reaches their computed lower bound of $\sim 1.475n \log n$. The concrete construction principle of both frameworks is the same.

Let us assume we aim to construct a $\Gamma_1(n)$ EUG $G = (V, E, P)$ for a circuit of size n with a k-way construction. First, we put k distinguished poles from the set P into a block called *superpole* that has k inputs and k outputs. Within this superpole, we can route edge-disjointly between its inputs and poles, and between its poles and outputs. In total, we have $\lceil n/k \rceil$ superpoles built by the poles set P. The k inputs and outputs of each superpole then can be used as poles for k instances of a $\Gamma_1(\lceil n/k \rceil - 1)$ nested EUG, which on a high level allows to find edge-disjoint paths between the superpoles of G.[7]

More formally, a superpole shall be able to edge-embed any so-called *augmented k-way block* (similar to augmented DAG in [32]). An augmented k-way block is a map that defines the routes between the inputs and poles of the superpole, and between poles and other poles and outputs.

Definition 5 (Augmented k-way Block). *An augmented k-way block $G = (V, E)$ for pole set P, superpole inputs I, and superpole outputs O is a directed graph such that*

- $V = P \cup I \cup O$, $P \cap I = P \cap O = \emptyset$ *and* $|I| = |O| = k$,
- $G[P] := (P, E^P)$ *has fanin/fanout 1,*
- $E = E^P \cup E^{io}$ *with* E^{io} *satisfying*
  - *(Soundness) Every* $e \in E^{io}$ *satisfies either* $e = (in, p)$ *or* $e = (p, out)$ *for* $p \in P, in \in I, out \in O$,
  - *(Completeness) For every source (resp. sink)* $p \in P$*, there exists at most one* $in \in I$ *(resp.* $out \in O$*) such that* $(in, p) \in E^{io}$ *(resp.* $(p, out) \in E^{io}$*).*

[6] $\sim 2.25n \log n$ when including the optimizations by Zhao et al. [50] for the 4-way construction.

[7] We distinguish between EUGs and nested EUGs as the recursively constructed nested EUGs differ from its first EUG in Liu et al.'s construction [32].

Algorithm 1: `Valiant`(P, k)

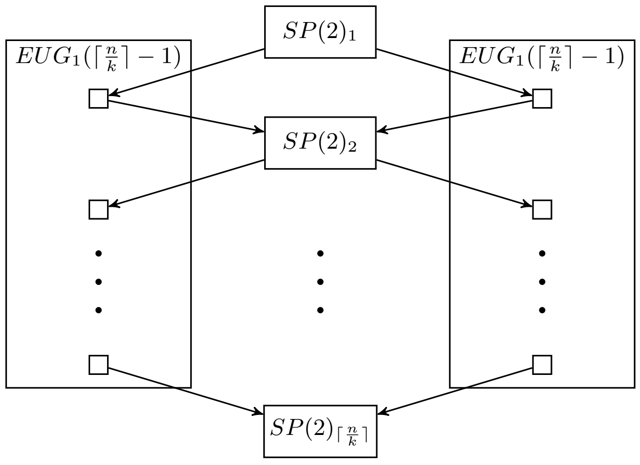
```
   Input  : Poles $P := \{p_1, ..., p_n\}$, split parameter $k$
   Output : $\Gamma_1(n)$ EUG $G = (V, E, P, G^*, G^1, ..., G^k)$
1  $V \leftarrow \emptyset,\ E \leftarrow \emptyset,\ G^* \leftarrow \emptyset$
2  $\mathcal{O}_0 \leftarrow$ create $k$ dummy nodes
3  for $i \leftarrow 1$ to $\lceil \frac{n}{k} \rceil$ do
4      $\mathcal{P}_i \leftarrow \{p_{k(i-1)+1}, ..., p_{ki}\}$
       // Use $\mathcal{O}_{i-1}$ as input recursion points to this superpole (cf.
          Fig. 3b)
5      $SP(k)_i = (V_i, E_i, P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i) \leftarrow$ Createsuperpole($\mathcal{P}_i, \mathcal{O}_{i-1}, k$);
        $G^* \leftarrow G^* \cup \{SP(k)_i\}$
6      $V \leftarrow V \cup V_i, E \leftarrow E \cup E_i$
7  for $i \leftarrow 1$ to $k$ do
8      if $n \leq k$ then
9          $G^i \leftarrow (\emptyset, ..., \emptyset)$ // Recursion base
10     else
           // Take the $i$-th output recursion point of each superpole
              (but the last) as the poles for the next sub EUG
11         $P^i \leftarrow \{\mathcal{O}_1[i], \mathcal{O}_2[i], ..., \mathcal{O}_{\lceil \frac{n}{k} \rceil - 1}[i]\}$
12         $G^i = (V^i, E^i, ...) \leftarrow$ Valiant($P^i, k$)
13         $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
14 return $G = (V, E, P, G^*, G^1, ..., G^k)$
```

The set of all augmented k-way blocks for P, I, O is denoted by $\mathcal{B}_k(P, I, O)$.

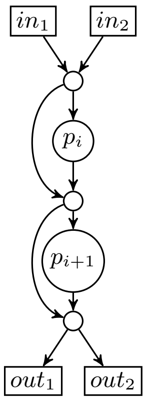
Definition 6 (k-way Superpole). *A k-way superpole, denoted by $SP(k)$ is a graph $G = (V, E, P, \mathcal{P}, \mathcal{I}, \mathcal{O})$, where the following conditions hold:*

- *$P = \mathcal{P} \cup \mathcal{I} \cup \mathcal{O}$ with $|\mathcal{I}| = |\mathcal{O}| = k$ and $\mathcal{P} \cap \mathcal{I} = \mathcal{P} \cap \mathcal{O} = \emptyset$.*
- *G can edge-embed every $G' \in \mathcal{B}_k(\mathcal{P}, \mathcal{I}, \mathcal{O})$.*

We denote the input recursion points $\mathcal{I}$ of a k-way superpole as $\{in_1, in_2, ..., in_k\}$ and the output recursion points $\mathcal{O}$ as $\{out_1, out_2, ..., out_k\}$. These nodes serve as the inputs and outputs to the superpole and will be the poles of the next recursion, i.e., of the next nested EUG. We neither require the sets $\mathcal{I}$ and $\mathcal{O}$ to be disjoint nor that the recursion points of different superpoles must be disjoint. In fact, Valiant [44] merges the output recursion points of the i-th superpole with the input recursion points of the $(i+1)$-th superpole. On a high level, a superpole in a nested EUG U^1, i.e., an EUG that is derived as a recursion from a larger EUG U, has k entry points to an input of k distinguished superpoles in U as well as k exit points from an output of k distinguished superpoles.


$EUG_1(\lceil\frac{n}{k}\rceil - 1)$
$SP(2)_1$
$EUG_1(\lceil\frac{n}{k}\rceil - 1)$
$SP(2)_2$
$SP(2)_{\lceil\frac{n}{k}\rceil}$

(a) Valiant's $EUG_1(n)$ construction.



(b) Valiant's $SP(2)$ construction.

Fig. 3: (a) shows Valiant's 2-way split construction of $EUG_1(n)$ using two instances of $EUG_1(\lceil\frac{n}{k}\rceil - 1)$. (b) shows the corresponding superpole $SP(2)$ construction for the EUG.

3.2 Valiant's EUG construction [44]

Definition 7 (Valiant EUG). *A Valiant EUG $G = (V, E, P, G^*, G^1, ..., G^k)$ is a graph that is created by Alg. 1 (*`Valiant`*). We also use the notation* $\texttt{Valiant}_k(n)$ *for a Valiant EUG with n poles and split parameter k.*

Valiant's k-way EUG construction is built recursively as depicted in Fig. 3a. A $\Gamma_1(n)$ EUG is a chain of $\lceil n/k\rceil$ superpoles $SP(k)_1 = (V_1,\ E_1,\ P_1,\ \mathcal{P}_1,\ \mathcal{I}_1,\ \mathcal{O}_1),\ \ldots,\ SP(k)_{\lceil n/k\rceil} = (V_{\lceil n/k\rceil},\ E_{\lceil n/k\rceil},\ P_{\lceil n/k\rceil},\ \mathcal{P}_{\lceil n/k\rceil},\ \mathcal{I}_{\lceil n/k\rceil},\ \mathcal{O}_{\lceil n/k\rceil})$ (lines 3-7 in Alg. 1). Note that $\texttt{Createsuperpole}(P, \mathcal{O}, k)$ creates a superpole with poles P, input recursion points $\mathcal{O}$, and split parameter k, e.g., Valiant's $k = 2$-way superpole $SP(2)$ (Fig. 3b). The sets $\mathcal{O}_1, \ldots, \mathcal{O}_{\lceil n/k-1\rceil}$ each of size k then recursively build the poles of the nested EUGs in the next recursion step (lines 8-14 in Alg. 1), i.e., we build k nested EUGs $G^1 = (V^1, E^1, P^1), \ldots, G^k = (V^k, E^k, P^k)$, where $G^i \in \Gamma_1(\lceil n/k\rceil - 1)$ and $P^i = (\mathcal{O}_1[i], \ldots, \mathcal{O}_{\lceil n/k-1\rceil}[i])$. Note that $\mathcal{I}_i := \mathcal{O}_{i-1}$ for all $1 < i \leq \lceil n/k\rceil$ as the k outputs of $SP(k)_i$ are pairwise merged with the respective k inputs of $SP(k)_{i+1}$. The creation of the first output recursion points $\mathcal{O}_0$ is a technical trick, and not needed because these nodes will never be used, but it simplifies the definition of the algorithm by avoiding a case distinction. An advantage of this recursive method is that we can also recursively reduce the edge-embedding problem to finding paths between poles of the nested EUGs. Assuming we can easily edge-embed paths from inputs to poles and from poles to outputs within the superpoles, one can reduce finding a path from a pole located in $SP(k)_i$ to a pole in $SP(k)_j$ to the problem of finding a path from $\mathcal{O}_i[x]$ to $\mathcal{O}_{j-1}[x]$ for $i, j \in [\lceil n/k\rceil]$, $i < j$, where x is the index of the target output of the superpoles' internal edge-embedding for the concrete poles. Existing UC implementations [19,3] split the edge-embedding into two sub-tasks: (a) the superpole edge-embedding that takes care that the paths within a superpole are defined in a correct manner, and (b) the recursion-point edge-embedding which chooses the correct paths at the recursion points, i.e., when a path goes one recursion step above or below.

Theorem 1. *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a Valiant EUG with n poles. Then G is an EUG for $\Gamma_1(n)$.*

We refer to [3,32] for a proof of Thm. 1.

3.3 Liu et al.'s EUG construction [32]

Definition 8 (Liu$^+$EUG). *A Liu$^+$EUG $G = (V, E, P, G*, G^1, \ldots, G^k)$ is a graph that is created by Alg. 2 (`Liu`$^+$). We also use the notation $\texttt{Liu}_k^+(n)$ for a Liu$^+$EUG with n poles and split parameter k.*

Liu et al.'s 2-way EUG construction as defined by Def. 8 is depicted in Fig. 4b and is separated into two part:

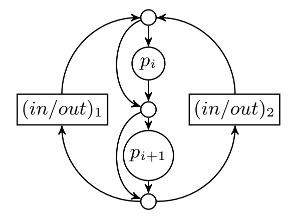
1. We create an intermediate construction that Liu et al. [32] call *weak EUG* (lines 1-13 in Alg. 2), which slightly differs from Valiant's construction but does not satisfy the acyclicness condition for EUGs (cf. Def. 3). This results in the graph depicted in Fig. 4b including the gray edges and nodes, but excluding the red edges.
2. We destroy the poles within the nested EUGs of the intermediate construction, which implicitly removes the cycles and leads to an EUG of smaller size (lines 14-27 in Alg. 2). This results in the graph depicted in Fig. 4b including the red edges, but excluding the gray edges and nodes.

As in Valiant's construction (cf. § 3.2), Liu et al. build their EUG as a chain of $\lceil n/k \rceil$ superpoles $SP(k)_1, \ldots, SP(k)_{\lceil n/k \rceil}$ (lines 3-7 in Alg. 2). For the nested EUGs, i.e., those EUGs that are built by recursion, we use the superpole depicted in Fig. 4a.[8] Recall, in Valiant's construction, we merge the input and output recursion points of neighboring superpoles, namely the i-th and the $(i+1)$-st superpoles. This time, however, we merge the input and output recursion points of each superpole individually as depicted in Fig. 4a, i.e., for $SP(k)_i = (V_i, E_i, P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i)$, the k nested EUGs in the next recursion step are built as $G^1 = (V^1, E^1, P^1), \ldots, G^k = (V^k, E^k, P^k)$, where $G^j \in \Gamma_1(\lceil n/k \rceil)$ and $P^j = (\mathcal{O}_1[j], \ldots, \mathcal{O}_{n/k}[j]) = (\mathcal{I}_1[j], \ldots, \mathcal{I}_{\lceil n/k \rceil}[j])$ for $i \in [\lceil n/k \rceil]$ and $j \in [k]$. As a consequence, this construction then yields a pole set of size $|P^i| = \lceil n/k \rceil$ (line 11 in Alg. 2) for the k nested EUGs as each superpole needs a separate input/output

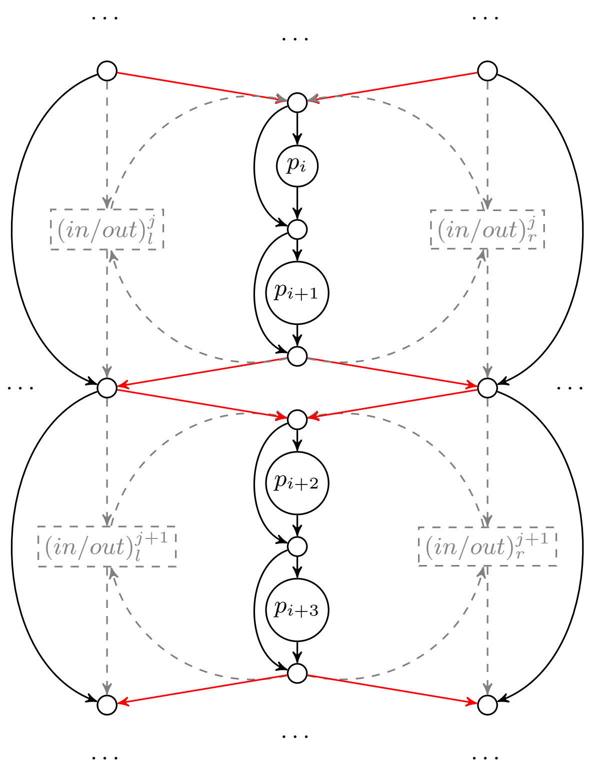
[8] Note that the main structure is equal to Valiant's superpole (cf. Figure 3b), but the input and output recursion points are merged. That is why `createSuperpole` in Alg. 2 does not need the second argument (cf. Alg. 1) as the recursion points of two superpoles are disjoint.

Algorithm 2: `Liu`$^+(P, k)$

```
Input  : Poles $P \coloneqq \{p_1, ..., p_n\}$, split parameter $k$
Output : $\Gamma_1(n)$ EUG $G = (V, E, P, G^*, G^1, ..., G^k)$
1  $V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $G^* \leftarrow \emptyset$
2  for $i \leftarrow 1$ to $\lceil \frac{n}{k} \rceil$ do
3      $\mathcal{P}^i \leftarrow \{p_{k(i-1)+1}, ..., p_{ki}\}$
4      $SP(k)_i = (V_i, E_i, P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i) \leftarrow$ Createsuperpole($\mathcal{P}_i, k$)
5      $G^* \leftarrow G^* \cup \{SP(k)_i\}$
6      $V \leftarrow V \cup V_i, E \leftarrow E \cup E_i$
7  for $i \leftarrow 1$ to $k$ do
8      if $n \leq k$ then
9          $G^i \leftarrow (\emptyset, ..., \emptyset)$ // Recursion base
10     else
           // Take the $i$-th output recursion point of each superpole as
              the poles for the next sub EUG
11         $P^i \leftarrow \{\mathcal{O}_1[i], \mathcal{O}_2[i], ..., \mathcal{O}_{\lceil \frac{n}{k} \rceil - 1}[i], \mathcal{O}_{\lceil \frac{n}{k} \rceil}[i]\}$
12         $G^i = (V^i, E^i, ...) \leftarrow$ Liu $^+P^i, k$
13         $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
14 foreach $(u, v) \in E$ do
15     if $u \in s$ and $v$ is recursion point for some superpole $s \in G^*$ then
16         $G^x \leftarrow$ the EUG in which $v$ is a pole
17         $E \leftarrow E \setminus \{(u, v)\}$
18         $w \leftarrow \Gamma^-_{G^x}(v)$
19         $E \leftarrow E \setminus \{(v, w)\}$
20         $E \leftarrow E \cup \{(u, w)\}$
21     else if $u$ is recursion point for some superpole $s \in G^*$ and $v \in s$ then
22         $G^x \leftarrow$ the EUG in which $u$ is a pole
23         $E \leftarrow E \setminus \{(u, v)\}$
24         $w \leftarrow \Gamma^+_{G^x}(u)$
25         $E \leftarrow E \setminus \{(w, u)\}$
26         $E \leftarrow E \cup \{(w, v)\}$
27 remove all recursion points from $V$
28 return $G = (V, E, P, G^*, G^1, \ldots, G^k)$
```



(a) 2-way-superpole.



(b) Basic structure for the 2-way split.

Fig. 4: (a) Superpole and (b) basic structure of Liu et al.'s 2-way split construction [32].

recursion point, while in Valiant's construction two neighboring superpoles are able to share one input/output recursion point yielding a pole set of size $|P_{\texttt{Val}^i}| = \lceil n/k \rceil - 1$ (line 12 in Alg. 1), i.e., Liu et al.'s intermediate construction even has a worse size than Valiant's construction. On top of that, as Fig. 4b shows, this merging of input and output recursion points within the same pole leads to cycles that are not allowed in EUGs according to Def. 3.

However, there is one key observation that allows to reduce the size of the intermediate EUG tremendously. Note, so far we have built the construction as depicted in Fig. 4b including the gray edges and excluding the red edges. Now, we want to remove the gray edges and introduce the red ones in order to get rid of the cycles and turn this weak EUG into a real EUG. First, let us have a look at the node $v = (in/out)_l^j$ in Fig. 4b and ignoring the red edges first. When we translate this graph into a UC, we would implement v as an X-switch because it has two inputs and two outputs. One input of this X-switch is the output of superpole $SP(k)_{i/2}$ (that contains the poles p_i and p_{i+1}) and one output of this X-switch is the input of the same superpole $SP(k)_{i/2}$. However, as we are not allowed to have cycles in a Boolean circuit that implements combinational logic, we are not allowed to program the X-switch such that its input coming from superpole $SP(k)_{i/2}$ is forwarded to its output that is directed to the input of superpole $SP(k)_{i/2}$. Consequently, the X-switch allows only one programming, namely the one that does not lead to a cycle. However, since the programming of this X-switch is fixed, we can replace it with two wires, i.e., we can remove the whole X-switch and thus also the node v and connect the corresponding wires to the other input and output of v that are not directed to/from the superpole $SP(k)_{i/2}$ (cf. red lines in Fig. 4b and lines 15-26 in Alg. 2). This reduces the size of the superpole in nested EUGs to 3 for $k = 2$ (resp. 14 for $k = 4$) as the two (resp. four) poles are removed.

Theorem 2 (cf. [32, Theorem 4]). *Let $G = (V, E, P, G^*, G^1, ..., G^k)$ be a Liu EUG with n poles. Then G is an EUG for $\Gamma_1(n)$ with size bounded by*

$$\frac{|SP(k)| - k}{k \log_2(k)} n \log_2(n) + \mathcal{O}(n).$$

We refer to [32] for the proof of Thm. 2.

4 Evaluating LUTs with UCs

In this section, we extend the UC constructions from § 3 to be able to simulate ($\rho \to \omega$)-LUT-based circuits. In § 4.1, we first review the construction of [44,29] to evaluate ($\rho \to 1$)-LUT-based circuits, i.e., circuits that consist of LUTs with ρ inputs and only one output. Afterwards, in § 4.2, we extend this to our LUT-based UCs (LUCs) that allows the UC to simulate ($\rho \to \omega$)-LUT-based circuits. Finally, in § 4.3, we analyze the most important building blocks for PFE applications, describe how to implement them with LUTs, and show their theoretical improvement over evaluating the same building blocks with Boolean circuits.

4.1 UCs for LUTs with multiple inputs [44,41]

Valiant [44] proposed a method to integrate LUTs with more than two inputs into UCs and its size has been computed in [41].

One can get a UC consisting of n copies of ($\rho \to 1$)-LUT from a $U = \Gamma_\rho(n)$ EUG that is merged by ρ instances of a $\Gamma_1(n)$ EUG according to Cor. 1. Each pole of U that is not an input or an output can then be implemented as a LUT having ρ inputs.

Corollary 2. *An EUG for $\Gamma_\rho(n)$ for $\rho \in \mathbb{N}_{\geq 2}$ can be constructed with size at most $1.5\rho n \log_2(n) + \mathcal{O}(n)$.*

Proof. Construct ρ instances of $\texttt{Liu(n)}_2^+$ and merge them. By Cor. 1, this yields an EUG for $\Gamma_\rho(n)$ with size bounded by $1.5\rho n \log_2(n) + \mathcal{O}(n)$. $\square$

4.2 UCs for LUTs with multiple in- and outputs

In order to support $(\rho \to \omega)$-LUTs with $\omega > 1$ outputs in UCs, we propose a general solution that is compatible with the original UC constructions of Valiant [44] and Liu et al. [32]. The high level idea is as follows: For every $(\rho \to \omega)$-LUT that is represented by pole v_i, we add $\omega - 1$ so-called *auxiliary* poles to the EUG and the real pole v_i forwards its inputs directly to these auxiliary poles. The real pole and its auxiliary poles each compute and output one of the LUT's output. Concretely, the first pole takes the ρ inputs of the LUT using any of the above UC construction and computes the first output of the LUT. The remaining poles copy the ρ inputs of the first poles by direct connections and compute the remaining outputs of the LUT. This results in a chain of ω poles each outputting one of the LUT's outputs.

We define the class of $\Gamma_{\rho,\omega}(n)$ graphs that is used to map n $(\rho \to \omega)$-LUTs to a graph $G \in \Gamma_{\rho,\omega}(n)$. As the poles of the EUG are the nodes of G, we need to add for each additional output of the i-th LUT in total $\omega - 1$ additional poles. We define $\Gamma_{\rho,\omega}(n)$ as follows:

Definition 9 ($\Gamma_{\rho,\omega}(n)$). *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V := \{v_{1,1}, \dots, v_{1,\omega}, v_{2,1}, \dots, v_{2,\omega}, \dots, v_{n,1}, \dots, v_{n,\omega}\}$ and $\rho, \omega \in \mathbb{N}$. Then $G \in \Gamma_{\rho,\omega}(n)$ if:*

- $|V| \leq n\omega$,
- $|\{v_{i,j} \in V\}| \leq \omega \forall i \in [n]$,
- $deg^+(v_{i,1}) \leq \rho \wedge deg^+(v_{i,2}) = \cdots = deg^+(v_{i,\omega}) = 0$,
- $deg^-(v_{i,j}) \leq \rho \;\forall\; i \in [n] \forall j \in [\omega]$.

To easily build an EUG with only marginal modifications, we show that $\Gamma_{\rho,\omega}(n)$ is also a $\Gamma_\rho(n\omega)$ graph:

Proposition 2. *Let $G \in \Gamma_{\rho,\omega}(n)$. Then $G \in \Gamma_\rho(n\omega)$.*

Proof. Let $G = (V, E) \in \Gamma_{\rho,\omega}(n)$. Obviously, it holds that $|V| \leq n\omega$ (condition 1 in Def. 9). Further, for all $v \in V$ it holds that $\deg^+(v) \leq \rho$ and $\deg^-(v_{i,j}) \leq \rho$ from condition 3 and 4 in Def. 9. Thus, $G \in \Gamma_\rho(n\omega)$. □

Now, we can build EUGs for multi-input and multi-output LUTs.

Corollary 3. *Let $\rho, \omega \in \mathbb{N}$. Then there exists a EUG for $\Gamma_{\rho,\omega}(n)$ with size bounded by*

$$1.5\rho n\omega \log_2(n\omega) + \mathcal{O}(n\omega)$$

Proof. **Step 1:** Create a $\Gamma_\rho(n\omega)$ EUG $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}}, P, \mathcal{U}^*, \mathcal{U}_1, \dots, \mathcal{U}_\rho)$ with a topologically ordered pole set P that has the form $(..., v_{i-1,\omega}, v_{i,1}, ..., v_{i,\omega}, v_{i+1,1}, ...)$ for all $i \in [n]$, i.e., the original pole $v_{i,1}$ directly preceding the auxiliary poles $v_{i,j}$ for $1 < j \leq \omega$:
We do this by creating a `Liu` EUG $\mathcal{U}$ with pole set P and split parameter 2.

Then we merge ρ instances of it. By Thm. 2 with $|SP(2)| = 5$ [32] and Cor. 1, this yields a $\Gamma_\rho(n\omega)$ EUG of size at most $1.5\rho n\omega \log_2(n\omega) + \mathcal{O}(n\omega)$.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}}, V, \bar{\mathcal{U}}^*, \bar{\mathcal{U}}_1, \ldots, \bar{\mathcal{U}}_\rho)$:
Let $v_{i,j}$ be an auxiliary pole of $v_{i,1}$ for $i \in [n], 1 < j \leq \omega$. Remove all of its incoming edges and replace each of them with an edge connecting the original pole $v_{i,1}$ with the auxiliary pole $v_{i,j}$, i.e., remove $(w, v_{i,j}) \in E^{\mathcal{U}}$ for $w \in V^{\mathcal{U}}$ and replace it by $(v_{i,1}, v_{i,j})$. This yields ρ edges $(v_{i,1}, v_{i,j})$ per auxiliary pole $v_{i,j}$ (one for each EUG instance). Thus, $E^{\mathcal{U}}$ becomes a multi set. The graph that results from modifying $\mathcal{U}$ in the just described way is denoted by $\bar{\mathcal{U}}$ and its pole set is denoted by V.

Step 3: Embed any graph $G = (V, E) \in \Gamma_{\rho,\omega}(n)$ $V := \{v_{1,1}, \ldots, v_{1,\omega}, v_{2,1}, ..., v_{n,\omega}\}$ into $\bar{\mathcal{U}}$:
To show that $\bar{\mathcal{U}}$ is a EUG for $\Gamma_{\rho,\omega}(n)$, we need to define an edge-embedding ψ from G into $\bar{\mathcal{U}}$. Thanks to Prop. 2, it holds that $G \in \Gamma_\rho(n\omega)$. Note that the "relative topological order" is maintained, i.e., $\eta_G(v_i) < \eta_G(v_{i+1})$ for $i \in [n]$. However, although $\bar{\mathcal{U}}$ has $n\omega$ poles, it is not an EUG for all $\Gamma_\rho(n\omega)$ graphs as all poles $v_{i,j>1}$ are directly connected to pole $v_{i,1}$ via the edge $(v_{i,1}, v_{i,j>1})$ for $i \in [n], j \in [\omega]$. Thus, we cannot find edge-disjoint paths from any pole $v_{k<i,l}$ to $v_{i,j>1}$ for $k \in [n]$, $l \in [\omega]$, as these would all use an ingoing edge of pole $v_{i,1}$. So, we need to show that all nodes $v_{i,j>1} \in G$ have indegree 0 to ensure that no edge-disjoint path needs to destinate at pole $v_{i,j>1} \in \bar{\mathcal{U}}$. This, however, is fulfilled due to condition 3 of Definition 9, i.e., there exists no edge $e = (v_{k<i,l}, v_{i,j>1}) \in G$ for which an embedding $\psi(e)$ needs to be defined (the same argument holds for edges $e = (v_{i,l<j}, v_{i,j}) \in G$).
So far, we showed that G only contains edges $e = (v_{k<i,l}, v_{i,1}) \in G$, which are the only ones to edge-embed into $\bar{\mathcal{U}}$. However, as we just added additional edges to poles $v_{i,1}$ and no outgoing edges from any poles in $\bar{\mathcal{U}}$ have been removed, we can get the edge-embedding ψ directly from Cor. 2. □

4.3 Improvement

In this section, we show the improvement of our LUC construction for several basic building blocks such as full adder (FA), comparator (CMP), and multiplexer (MUX). As summarized in Tab. 2, our basic building blocks are smaller than the previous constructions [19,27,32,3] in UC size by factor 2× - 2.7×. Note that we compute the improvement factors using only the prefactor, so concrete improvements will be larger as also the logarithmic term is improved. This UC size reduction is achieved by merging smaller 2-input gates into larger multi-input look-up-tables (LUT).
Full Adder (FA): The optimized implementation of a FA uses four 2-input XOR gates and one 2-input AND gate (cf. [28, Fig. 2]). We can implement a FA using only one $(3 \rightarrow 2)$-LUT, resulting in an improvement by $\geq 2\times$ in LUC size (cf. Tab. 2).
Comparator (CMP): The 1-bit comparator consists of three 2-input XOR gates and a single 2-input AND gate (cf. [28, Fig. 6]). Our improved LUT-based instantiating for CMP uses only one $(3 \rightarrow 1)$-LUT, resulting in an improvement

of $\geq 2.7\times$ in LUC size (cf. Tab. 2).
Multiplexer (MUX): The MUX block can be instantiated with one 2-input AND gate and two 2-input XOR gates (cf. [30, Fig. 2]). In our approach, MUX can be instantiated with only one $(3 \rightarrow 1)$-LUT, resulting in an improvement of $\geq 2\times$ in the LUC size (cf. Tab. 2).

Complex Building Blocks. We now present several motivating examples that benefit from improvements of our basic building blocks.
Addition and Subtraction. An l-bit addition is composed of a chain of l Full Adders (FA) (cf. [28, Fig. 1]. An l-bit subtraction is defined as $x - y = x + y + 1$ and can be constructed similarly to an addition circuit using l FAs (cf. [28, Fig. 3]. Using our FA construction, the LUC size of the addition and subtraction is improved by $\geq 2\times$.
Multiplication. Multiplication of two l-bit numbers can be composed of l^2 of 2-input AND gates ($(2 \rightarrow 1)$-LUT) and $(l-1)$ l-bit adders [28]. Using the efficient implementation for LUT-based adders, the LUC size of the multiplication circuit is improved by $\geq 2\times$.
Multiplexer. An l-bit multiplexer circuit can be composed of l parallel MUX (cf. [30, Fig. 9]) to select one of the l-bit inputs. So, using our LUT-based MUX has $\geq 2\times$ improvement for an l-bit multiplexer.
Comparison. An l-bit comparison circuit can be composed of a chain of l CMPs (cf. [28, Fig. 5]). Thus, deploying our CMP construction improves the LUC size of the comparison circuit by $\geq 2.7\times$. A minimum circuit which selects the minimum value of a list of m l-bit values is composed of l-bit comparison and multiplexer circuits (cf. [28, Fig. 8]) and hence is improved by $\geq 2.3\times$.

Table 2: LUC sizes for basic building blocks which can be used to construct more complex functionalities.

Building Block (BB)	Boolean Circuit		LUT-based Circuit		Improvement
	# Gates	Asympt. UC Size	LUT type	Asympt. LUC Size	
FA	4 XOR 1 AND	$15n \log 5n + \mathcal{O}(n)$	$(3 \rightarrow 2)$-LUT	$7.5n \log 2.5n + \mathcal{O}(n)$	$\geq 2\times$
CMP	3 XOR 1 AND	$12n \log 4n + \mathcal{O}(n)$	$(3 \rightarrow 1)$-LUT	$4.5n \log 1.5n + \mathcal{O}(n)$	$\geq 2.7\times$
MUX	2 XOR 1 AND	$9n \log 3n + \mathcal{O}(n)$	$(3 \rightarrow 1)$-LUT	$4.5n \log 1.5n + \mathcal{O}(n)$	$\geq 2\times$

5 Our Varying UC (VUC) Construction

The size improvements of our LUT-based UC (LUC) construction (cf., § 4.3) are most significantly for LUT sizes of $\rho = 3$ inputs for the majority of functions. For larger input arities $\rho > 4$, LUC might even perform worse than the baseline of circuits with $\rho = 2$.[9] This highly depends on the function: the design of LUC requires to merge ρ instances of $\Gamma_1(n)$ EUGs, which allows the usage of n $(\rho \to \omega)$-LUTs, and the size of the LUC asymptotically grows superlinearly with ρ. However, in most cases, only a few sub-functionalities of the function profit from the usage of higher LUT arities and the potential of most $\rho > 4$ input LUTs is not used. In many applications, functionalities are naturally implemented by LUTs with higher arity, e.g., Sboxes in AES. In this case, we aim to put single LUTs with a higher arity (e.g., $\rho = 8$) into the UC. Using our LUC constuction for this concrete example, we would need to compose the UC of 8 instances of $\Gamma_1(n)$ EUGs, even if we only need the full $(8 \to 1)$-LUT few times in the whole circuit.[10] Thus, our aim is to find a way to use single LUTs with input arity of $\rho > 4$ without a massive influence on the total circuit size.

In this section, we present our Varying UC (VUC) construction, which deviates from the conventional universal circuits (UCs) that have been widely studied [44,29,27,31,19,3,32]. Traditionally, UCs have been designed to conceal both the topology and the gate functionality of the simulated function, and have relied on the use of fixed computational units, namely universal 2-input gates or, like in our work, $(\rho \to \omega)$-LUTs with a globally fixed number of inputs ρ and outputs ω. A VUC, however, allows for the use of different programmable computational units, thereby leaking information about the types of units used. In particular, we focus on VUCs built using $(\rho \to \omega)$-LUTs with varying numbers of inputs and outputs, thereby revealing the arity of the individual LUTs.

Definition 10 (Varying Universal Circuit (VUC)). *A Varying Universal Circuit $\mathcal{V}$ for n_i inputs, the set of n_g gates $G = \{G_1, \ldots, G_{n_g}\}$ of varying input and output arities, and n_o outputs is a Boolean circuit that can be programmed to compute any Boolean circuit C with n_i input and n_o outputs that contains a subset of the gates in G by defining a set of programming bits p^C such that $\mathcal{V}(x, p^C) = C(x)$ for all possible input values $x \in \{0,1\}^{n_i}$.*

In § 5.2, we discuss several applications of VUCs as well as their leakage.

5.1 The VUC construction

First, we show how to build our VUC for evaluating different $(\rho \to 1)$-LUT with varying LUT input arities ρ. Later in this section, we show how to extend this construction to evaluate any $(\rho \to \omega)$-LUTs with varying input and output arities ρ and ω. In our VUC construction, we keep building our UC from only two instances of a $\Gamma_1(n)$ EUG, independent of the LUT sizes. This reduces the overhead of our LUT-based UC construction that merges ρ instances of the large $\Gamma_1(n)$ EUG for $(\rho \to 1)$-LUT. We do this by adding auxiliary poles u to the EUG whose task is to collect up to two inputs and forward these inputs via direct edges

[9] However, LUC with $\rho = 3$ input LUTs performs always better than UCs for Boolean circuits from a certain function size n on.

[10] An alternative would be to decompose the larger LUTs into multiple smaller ones using Shannon expansion [42].

Algorithm 3: `AuxiliaryGraph`(G)

```
Input  : $G = (V, E) \in \mathit{\Gamma}_{\mathrm{P}^+,2}(n)$
Output : $\bar{G} = (\bar{V}, \bar{E}) \in \mathit{\Gamma}_2(n + \mathit{\Delta})$ with $\mathit{\Delta} = \sum_{i=0}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$
1  $\bar{G} = (\bar{V}, \bar{E}) \leftarrow (V, \emptyset)$
2  foreach $v_i \in V$ do
3      $j \leftarrow 0$
4      foreach $e = (w, v_i) \in E$ do
5          if $j \geq 2$ then
6              if $j \equiv 0 \pmod 2$ then
7                  $\bar{V} \leftarrow \bar{V} \cup \{u_{i,\frac{j}{2}}\}$
8              $\bar{E} \leftarrow \bar{E} \cup \{(w, u_{i,\lceil \frac{j}{2} \rceil})\}$
9          else
10             $\bar{E} \leftarrow \bar{E} \cup \{e\}$
11         $j \leftarrow j + 1$
```

to a real pole v to push the indegree of v to ρ. Def. 11 defines $\mathit{\Gamma}_{\mathrm{P}^+,\mathrm{P}^-}(n)$ graphs, which classify the graphs that can be edge-embedded into our VUC construction, namely, the vectors P^+ and P^- specify the maximum indegree and outdegree of each LUT in our circuit that we aim to evaluate with the UC. Our VUC design additionally allows the evaluation of functions that only use a single type of ρ input LUTs by setting $\mathrm{P}^+ = \mathbb{1}\rho$,[11] i.e., each LUT in the circuit can have at most ρ inputs and the resulting VUC implements each universal gate as a ($\rho \rightarrow 1$)-LUT. In this case, the VUC is a real LUT-based UC (LUC) and can be used for PFE. In the case for VPFE, the universal gates of the UC have different implementations and therefore leak the specific input sizes of all LUTs.

Definition 11 ($\mathit{\Gamma}_{\mathrm{P}^+,\mathrm{P}^-}(n)$). *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V := \{v_1, ..., v_n\}$ and $\mathrm{P}^+, \mathrm{P}^- \in \mathbb{N}^n$. Then $G \in \mathit{\Gamma}_{\mathrm{P}^+,\mathrm{P}^-}(n)$ if:*

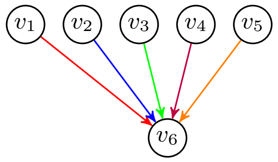
- $|V| \leq n$,
- $deg^+(v_i) \leq \mathrm{P}_i^+ \wedge deg^-(v_i) \leq \mathrm{P}_i^- \;\; \forall\, i \in [n]$.

If $\mathrm{P}^{+/-} = \mathbb{1}\rho$ for some $\rho \in \mathbb{N}$, we write ρ instead of $\mathbb{1}\rho$.

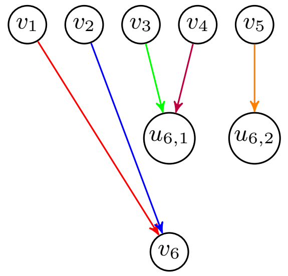
In this sense, Corollary 2 yields a $\mathit{\Gamma}_{\rho,\rho}(n)$ EUG. In the following, we describe our VUC construction. An example of the whole EUG creation and embedding process is depicted in Fig. 5. The explicit creation of the used auxiliary graph is given by Alg. 3.

The key observation for our VUC construction is that, when merging two instances of $\mathit{\Gamma}_1(n)$ EUGs, each of the n poles (excluding inputs and outputs) can take two inputs, and *can*, but not necessarily need to, compute one output.

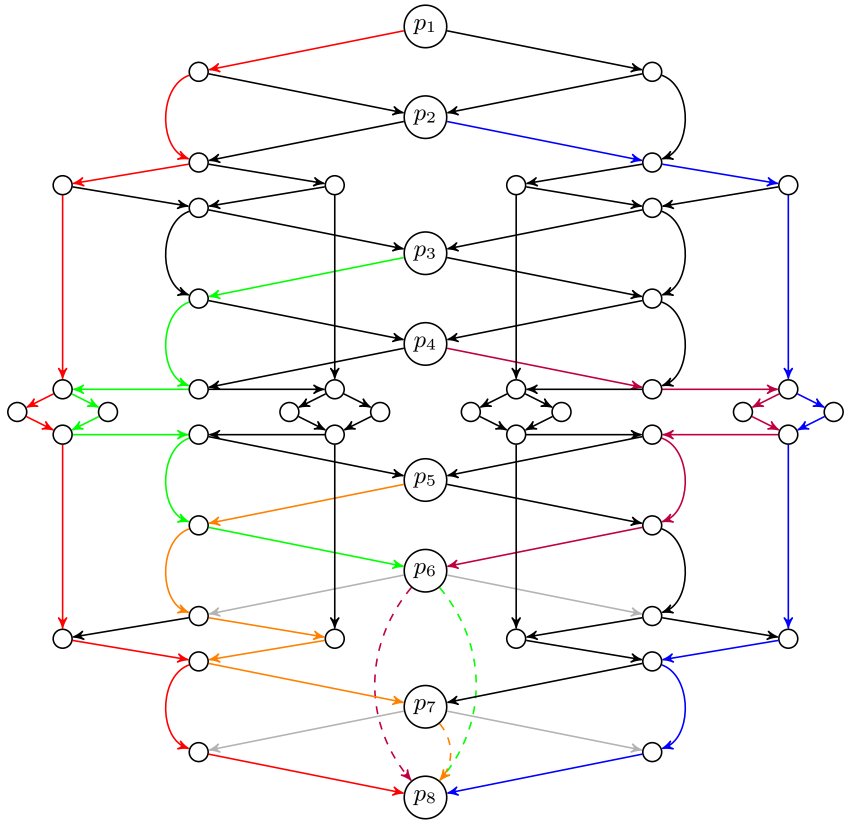
[11] $\mathbb{1}$ denotes the vector where each entry is 1.



(a) Original graph



(b) Corresponding auxiliary graph



(c) Edge-embedding of the original graph. First, the edges from the auxiliary graph are embedded. Then, edges in gray are removed from the EUG, while dashed edges are added to the EUG, resp. to the edge-embedding. The result is an edge-embedding for the original graph. Now we can replace the ingoing edges to p_6 by directed edges to the multi-input pole p_8. The auxiliary pole p_7 becomes a Y-Switch that only forwards the orange wire.

Fig. 5: Our varying UC construction for $\rho = 5$.

We can use this observation to merge poles in order to collect $\rho > 2$ inputs for our LUT. For example, looking at Fig. 5, a $(5 \to 1)$-LUT consists of the three poles p_6, p_7, and p_8, where pole p_6 (resp. p_7) just collects two (resp. one) inputs, but do not compute any output. Instead, their ingoing edges are forwarded to pole p_8 (dashed lines) and the outgoing edges (gray lines) are removed. Pole p_8 now has, in addition to its two regular ingoing edges, three additionel ingoing edges that come directly from poles p_6 and p_7. On a high level, we can merge $\lceil \rho/2 \rceil$ poles into one $(\rho \to 1)$-LUT, while the first $\lceil \rho/2 \rceil - 1$ so-called *auxiliary poles* each collect up to two inputs for the LUT which are then directly forwarded to the last pole, which takes the last two inputs of the LUT and computes the output.

More formally, we begin by constructing an auxiliary graph $\bar{G}$. For each pole p that has $\rho > 2$ incoming edges, we create an auxiliary pole for each two additional inputs, i.e., $\lceil \rho/2 - 1 \rceil$ auxiliary poles. Then, we replace all except two edges from pole p by edges to the auxiliary poles. The purpose of the auxiliary poles is to forward their inputs to the original multi-input pole. The resulting EUG $\mathcal{U}$ then guarantees that there can be a path from any pole with lower order to the corresponding auxiliary poles.

If there is a multi-input gate with an odd number of inputs ρ, then there will be one auxiliary pole in $\bar{G}$ with only one input. In this case, we can share this auxiliary pole for two poles if both have an odd number of inputs (which is always the case in the special case of PFE). This concrete auxiliary pole is then later translated into an X-switching block so that the inputs can be forwarded to the correct LUT.

Theorem 3. *Let* $\mathrm{P}^+ \in \mathbb{N}^n$*. Then there exists an EUG for* $\Gamma_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3(n+\Delta)\log_2(n+\Delta) + \mathcal{O}(n+\Delta),$$

where $\Delta := \sum_{i=1}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$.

Proof. **Step 1:** Create a $\Gamma_2(n+\Delta)$ EUG $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}}, P, \mathcal{U}^*, \mathcal{U}_1, \mathcal{U}_2)$ with a topologically ordered pole set P that has the form $(..., v_{i-1}, u_{i,1}, ..., u_{i,\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil}, v_i, ...)$ for all $i \in [n]$, i.e., the auxiliary poles $u_{i,j}$ for $j \in [\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil]$ are directly preceding the original pole v_i:
We do this by creating a `Liu` EUG $\mathcal{U}$ with pole set P and split parameter 2. Then we merge two instances of it. By Thm. 2 and Cor. 1, this yields a $\Gamma_2(n+\Delta)$ EUG of size at most $3(n+\Delta)\log_2(n+\Delta) + \mathcal{O}(n+\Delta)$.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}}, V, \bar{\mathcal{U}}^*, \bar{\mathcal{U}}_1, \bar{\mathcal{U}}_2)$:
Let $u_{i,j}$ be an auxiliary pole of v_i for $i \in [n], j \in [\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil]$. Remove all of its outgoing edges and replace each of them with an edge connecting the auxiliary pole to the original multi-input pole, i.e., remove each $(u_{i,j}, w) \in E^{\mathcal{U}}$ for $w \in V^{\mathcal{U}}$ and replace it by $(u_{i,j}, v_i)$. This yields two edges $(u_{i,j}, v_i)$ per auxiliary pole $u_{i,j}$. Thus, $E^{\mathcal{U}}$ becomes a multi set.
If P_i^+ is odd and $j = 1$, add only one of these edges instead of two (otherwise, v_i

would have too many ingoing edges). The graph that results from modifying $\mathcal{U}$ in the just described way is denoted by $\bar{\mathcal{U}}$.

Step 3: Embed any graph $G = (V, E) \in \Gamma_{\mathrm{P}^+,2}(n)$ $V := \{v_1, v_2, ..., v_n\}$ into $\bar{\mathcal{U}}$: For this, we construct a $\Gamma_2(n + \Delta)$ graph using auxiliary poles for nodes with indegree higher than 2 by setting $\bar{G} = (\bar{V}, \bar{E}) =$ `auxiliaryGraph(G)` $\in \Gamma_2(n+\Delta)$ (Alg. 3). Note that the "relative topological order" is maintained, i.e., $\eta_{\bar{G}}(v_i) < \eta_{\bar{G}}(v_{i+1})$ $\forall i \in [n]$. Edge-embedding $\bar{G}$ into $\bar{\mathcal{U}}$ yields $\psi \colon \bar{E} \to \mathcal{P}_{\bar{\mathcal{U}}}$. To show that $\bar{\mathcal{U}}$ is a $\Gamma_{\mathrm{P}^+,2}(n)$ EUG, we need to define an edge-embedding $\bar{\psi}$ from G into $\bar{\mathcal{U}}$:
Note that for edges $e = (v_i, v_l) \in G \setminus \bar{G}$, i.e., edges whose endpoints are not auxiliary poles, ψ already yields edge-disjoint v_i-v_l-paths and we can set $\bar{\psi}(e) = \psi(e)$ for those edges.
Now consider edges $e = (v_i, v_l) \in G \cap \bar{G}$, i.e., the endpoints of those edges are transformed into an auxiliary pole in $\bar{G}$. For each e, there is exactly one $\bar{e} = (v_i, u_{l,j}) \in \bar{G}$ for $j \in [\lceil \frac{\deg^+(v_l)-2}{2} \rceil]$ (line 8 in Alg. 3). Now set $\bar{\psi}(e) = \psi(\bar{e}) + (u_{l,j}, v_l)$ for one of the possibly two edges $(u_{l,j}, v_l)$ that were added to $\bar{\mathcal{U}}$ before. Obviously, this yields a v_i-v_l-path. Since there are at most two edges connecting to an auxiliary pole, we can choose a unique last edge for each path. Because the paths in the image of ψ were already edge-disjoint, also the paths in the image of $\bar{\psi}$ are edge-disjoint. Thus, $\bar{\psi}$ is an edge-embedding of G into $\bar{\mathcal{U}}$. □

Thm. 3 gives us an EUG that can be used to build VUCs and can thus be used for Varying Private Function Evaluation (VPFE). Next, we consider LUCs for circuits consisting of $(\rho \to 1)$-LUTs only which yields classical PFE.

Corollary 4. *Let* $\mathrm{P}^+ = \mathbb{1}\rho \in \mathbb{N}^n$ *for* $\rho > 2$. *Then there exists a EUG for* $\Gamma_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3\lceil \frac{\rho}{2} n \rceil \log_2(\lceil \frac{\rho}{2} n \rceil) + \mathcal{O}(\lceil \frac{\rho}{2} n \rceil).$$

Proof. Following the proof of Thm. 3, we do the same steps and highlight the differences.

Step 1: Create a $\Gamma_2(\lceil \frac{\rho}{2} n \rceil)$ EUG $\mathcal{U}$ with topologically ordered pole set P that has the form $(..., v_{i-1}, u_{i,1}, ..., u_{i,\lceil \frac{\rho-2}{2} \rceil}, v_i, u_{i+1,1}, ..., u_{i+1,\lfloor \frac{\rho-2}{2} \rfloor}, v_{i+1}, ...)$ as described in step 1 in the proof of Thm. 3.

Step 2: Adjust $\mathcal{U}$ to get the final EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}}, V, \bar{\mathcal{U}}^*, \bar{\mathcal{U}}_1, \bar{\mathcal{U}}_2)$ as described in step 2 in the proof of Thm. 3 with one difference: If ρ is odd, we share one auxiliary pole $u_{i,1}$ for two consecutive original poles v_i and v_{i+1}, i.e., we add the two edges $(u_{i,1}, v_i)$ and $(u_{i,1}, v_{i+1})$.

Step 3: Edge-embed G into $\bar{\mathcal{U}}$ as described in step 3 in the proof of Thm. 3 with one difference: If ρ is odd, the auxiliary graph $\bar{G} = (\bar{V}, \bar{E})$ shares on auxiliary pole $u_{i,1}$ for two consecutive original poles v_i and v_{i+1}, i.e., $u_{i+1,1}$ is removed from $\bar{V}$ and the edge $(w, u_{i+1,1})$ is replaced by the edge $(w, u_{i,1})$. As $u_{i,1}$ and $u_{i+1,1}$ both have indegree 1, $u_{i,1}$ now has indegree 2. □

Multi-Output support for VUCs. An auxiliary graph that represents multi-output LUTs is a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$ graph as defined in Def. 12, i.e., $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$

classifies the graphs that can be edge-embedded into our UC construction. Here, P^+ is a vector of size n that specifies the indegree of each node in the auxiliary graph and thus represents the maximum number of inputs of each LUT in the UC. P^- is a constant that specifies the maximum outdegree of each node in the auxiliary graph / of each LUT in our circuit that we aim to evaluate with the UC. Similarly, Ω^- describes the number of distinguished outputs of the LUTs, i.e., P^- specifies the number of copies we have for each output of a LUT in our circuit, while Ω^- sets the number of outputs for each LUT.

As later, when embedding G into the EUG, each output of a LUT represents a separate value, i.e., we need to put each output into an individual pole. As the poles of the EUG are the nodes of the auxiliary graph, we need to add for each additional output of the i-th LUT in total $\Omega_i^- - 1$ additional poles. In Def. 12, we denote the outputs of the i-th LUT with $v_{i,1}, \ldots, v_{i,\Omega_i^-}$.

Definition 12 ($\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$). *Let $G = (V, E)$ be a directed acyclic graph with topologically ordered $V \coloneqq \{v_{1,1}, \ldots, v_{1,\Omega_1^-}, v_{2,1}, \ldots, v_{2,\Omega_2^-}, \ldots, v_{n,1}, \ldots, v_{n,\Omega_n^-}\}$ and $\mathrm{P}^+, \mathrm{P}^-, \Omega^- \in \mathbb{N}^n$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$ if:*

- $|V| \leq \sum_{i=1}^{n} \Omega_i^-$,
- $|\{v_{i,j} \in V\}| \leq \Omega_i^- \forall i \in [n]$,
- $deg^+(v_{i,1}) \leq \mathrm{P}_i^+ \wedge deg^+(v_{i,2}) = \cdots = deg^+(v_{i,\Omega_i^-}) = 0$,
- $deg^-(v_{i,j}) \leq \mathrm{P}_i^- \ \forall\ i \in [n] \forall j \in [\Omega_i^-]$.

To easily build an EUG with only marginal modifications, we show that a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}$ is also a $\Gamma_{\mathrm{P}^+,\mathrm{P}^-}$ graph:

Proposition 3. *Let $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$. Then $G \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-}(n + \Delta)$, where $\Delta \coloneqq \sum_{i=1}^{n} \Omega_i^- - 1$.*

Proof. Let $G = (V, E) \in \Gamma_{\mathrm{P}^+,\mathrm{P}^-,\Omega^-}(n)$. Obviously, it holds that $|V| \leq \sum_{i=1}^{n} \Omega_i^- = n + \Delta$ where $\Delta = \sum_{i=1}^{n} \Omega_i^- - 1$ (condition 1 in Def. 12). Further, for all $v \in V$ it holds that $\deg^+(v) \leq \mathrm{P}_i^+$ and $\deg^-(v_{i,j}) \leq \mathrm{P}_i^-$ from condition 3 and 4 in Def. 12. $\square$

Now, we can build VUCs using Cor. 5, as well as UCs where all LUTs have the same number of inputs and outputs using Cor. 6.

Corollary 5. *Let $\mathrm{P}^+, \Omega^- \in \mathbb{N}^n$. Then there exists a EUG for $\Gamma_{\mathrm{P}^+,2,\Omega^-}(n)$ with size bounded by*

$$3(n + \Delta)\log_2(n + \Delta) + \mathcal{O}(n + \Delta),$$

where $\Delta \coloneqq \sum_{i=1}^{n} (\max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\} + \Omega_i^- - 1)$.

Proof. Let $G = (V, E) \in \mathit{\Gamma}_{\mathrm{P}^+,2,\mathit{\Omega}^-}(n)$ be the graph to be embedded in an EUG with $V = \{v_{1,1}, \ldots, v_{1,\mathit{\Omega}_1^-}, v_{2,1}, \ldots, v_{2,\mathit{\Omega}_2^-}, \ldots, v_{n,1}, \ldots, v_{n,\mathit{\Omega}_n^-}\}$. We can transform G into a $\mathit{\Gamma}_{\mathrm{P}^+,2}(n + \mathit{\Delta}')$ graph where $\mathit{\Delta}' := \sum_{i=1}^{n} \mathit{\Omega}_i^- - 1$. Using Thm. 3, we get an EUG for $\mathit{\Gamma}_{\mathrm{P}^+,2}(n + \mathit{\Delta}')$ that is bounded by

$$3(n + \mathit{\Delta}' + \mathit{\Delta}'') \log_2(n + \mathit{\Delta}' + \mathit{\Delta}'') + \mathcal{O}(n + \mathit{\Delta}' + \mathit{\Delta}''),$$

where $\mathit{\Delta}'' := \sum_{i=1}^{n} \max\{\lceil \frac{\mathrm{P}_i^+ - 2}{2} \rceil, 0\}$ and $\mathit{\Delta} := \mathit{\Delta}' + \mathit{\Delta}''$ follows.

We need to add some more edges to the resulting EUG $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}}, V, \bar{\mathcal{U}}^*, \bar{\mathcal{U}}_1, \bar{\mathcal{U}}_2)$ with pole set V, namely the inputs of the first pole associated with the LUT need to be forwarded to all remaining output poles of the same LUT as follows:

$\forall i \in [n] : \forall v_{i,1} \in V : \forall (u, v_{i,1}) \in E^{\bar{\mathcal{U}}} : \forall v_{i,j} \in V, j > 1 : E^{\bar{\mathcal{U}}} = E^{\bar{\mathcal{U}}} \cup (u, v_{i,j})$. □

Corollary 6. *Let* $\mathrm{P}^+ = \rho \in \mathbb{N}^n$ *for* $\rho > 2$ *and* $\mathit{\Omega}^- = \omega \in \mathbb{N}^n$ *for* $\omega > 1$. *Then there exists an EUG for* $\mathit{\Gamma}_{\mathrm{P}^+,2}(n)$ *with size bounded by*

$$3(\lceil(\frac{\rho}{2} + \omega - 1)n\rceil) \log_2(\lceil(\frac{\rho}{2} + \omega - 1)n\rceil) + \mathcal{O}(\lceil(\frac{\rho}{2} + \omega - 1)n\rceil).$$

Proof. Follows directly from Cor. 4 and Cor. 5. □

5.2 Applications of Varying UCs (VUCs)

If we use a VUC instead of a UC in an MPC-based PFE protocol, we get Varying Private Function Evaluation (VPFE). VPFE allows a set of k parties $\mathcal{P}_1, \ldots, \mathcal{P}_k$, to jointly compute a *varying private* circuit C held by $\mathcal{P}_1$ on private data $x_2, \ldots, x_k$ held by $\mathcal{P}_{i \geq 2}$ to obtain nothing but $C(x_2, \ldots, x_k)$, and $\mathcal{P}_{i \geq 2}$ learn nothing about C but the arities of all its LUTs. Thus, VPFE does not leak the topology of complete sub-circuits like SPFE, but still leaks more information than PFE.

We can weak down the leakage by non-deterministically change the sequence of LUTs according to the topological order of the simulated circuit. In this way, building blocks (e.g., full adders) are not directly visible as a whole block in the VUC. In this way, the function would be mapped to different sequences of arities and thus we would remove fingerprints of certain functions. So, even multiple building blocks of different circuit layers can be mixed in a sequence. This technique, however, still allows to exclude certain functions when they cannot be mapped to the given sequence of arities.

Some applications, such as logic locking (cf., [7, Fig. 3]) do not require full privacy of the evaluated function and allow for the leakage of the sequence of arities of the used LUTs. However, in general PFE applications, even knowledge of the LUT sizes may reveal too many information about the protected function. Our analysis (cf., § 4.3) and our benchmarks (cf., § 6.2) demonstrate that LUCs offer the most size benefit when the circuit consists of $(3 \rightarrow \omega)$-LUTs, i.e., many functionalities can be reduced to 3-input LUTs. Thus, in most cases, we would benefit from using LUCs with 3-input LUTs. This observation is not surprising, as most arithmetic operations can be reduced to full adders (3-input LUTs), and only a small number of sub-functionalities benefit from using LUTs with more than 3 inputs. However, when adding only one of these larger LUTs, the overall size of the LUC would be significantly increased, as a complete EUG graph would need to be added to the circuit for each additional input of all LUTs, even if the higher arity is used only once. Many PFE applications, such as credit checking [14], user-specific tariff calculations [18], and medical diagnosis [5], rely on sub-functionalities such as classifiers. A classifier utilizes a database to look up a class based on input data, and then outputs the determined class. To illustrate, a car insurance tariff calculator may use a classifier to establish a basic price based on the type of car a potential customer drives. Multi-input LUTs can efficiently implement these classifiers as they operate similarly to database lookups. By incorporating individually tailored multi-input LUTs in a VUC, we can benefit from overall size improvements over the LUC construction, while still maintaining the internal implementation of the classifier, including the data it uses as reference as the topology is hidden.

6 Implementation and Evaluation

We implement our proposed UC constructions using the MPC framework ABY [10]. We benchmark our LUC construction (cf. §4) and compare it with the most recent UC of Liu et al [32] that simulates circuits with binary gates. Moreover, we evaluate our VUC construction (cf. §5) to show the improvement over today's most efficient of [32] and our LUC construction. All results in this section use EUG construction by Liu et al. [32] to construct the underlying Γ_1 EUGs. We discuss the LUT generation in §6.1, and experimental results in §6.2. Details about our UC compilation are provided in App. B.

6.1 LUT Generation

Moving away from 2-input Boolean gates requires LUT-based circuit representations of functions. Engineering and modifying hardware synthesis tools beyond their intended uses and adapting their output to our LUT-based UC representation required significant engineering. To address this, we provide a toolchain that converts high-level function descriptions into LUT representations. Similar to [11, §5], we use hardware synthesis tools, such as Yosys [46] and the ABC mapping [6], to automatically and effectively generate multi-input/multi-output LUT representations. ABC [6] does not, by default, support mapping to multi-output LUTs. To address this limitation, we post-process the single-output LUT circuits produced by ABC [6] and convert them to multi-output LUT circuits. We further use integrated Intellectual Property (IP) libraries in the Synopsys Design Compiler (DC) [1], a commercial ASIC synthesis tool, to generate circuit netlists for more complicated functionalities, such as floating-point operations. Synopsys DC creates these circuits as Boolean netlists, which we then remap to LUT-based representations using the Yosys-ABC toolchain.

6.2 Experimental Results

Setup. Like previous works [19,27,32], we benchmark a set of real-world circuits from [43]. In addition, we consider other useful functions like Karatsuba multiplication [24], Manhattan and Euclidean distance [11], and floating-point operations [11]. For each functionality, we give the sizes of the resulting circuit, as well as communication and runtime complexity when the UC is evaluated with an MPC protocol. In order to show the improvement of our work, we use two identical machines with a LAN connection of 10 Gbit/s bandwidth and a round-trip time of 1 ms. Each machine is equipped with an Intel Core i9-7960X@2.8 GHz with 128GB DDR4 RAM. All measurements are averaged over 10 executions.

Table 3: Number of AND and XOR gates per building block in our UCs.

Building block	AND gates	XOR gates
X-Switching block [30]	1	3
Y-Switching block [30]	1	2
Universal Gate with $k \geq 2$ inputs	$2^k - 1$	$2^{k+1} - 2$

LUC Improvement. As we have Universal Gates of different sizes, we cannot just count the number of nodes in the EUG to compare the implementations. Therefore, we count the number of AND gates that are necessary to instantiate the building blocks of the UC (cf. Table 3). As underlying MPC protocol for UC-based PFE we use Yao's protocol [49] using free XORs [30], so XOR gates can be evaluated without communication. We experimentally compared our implementations with the best existing UC-based PFE construction of [32]. We provide our results for our LUT-based UC constructions in Tab. 4. In our circuit generation, we vary possible choices for $(\rho \rightarrow \omega)$-LUTs and select the ones with highest improvement. We can see from Tab. 4 that our LUT-based UC construction is always smaller than that of Liu et al.'s [32] by $1.13 - 2.18\times$.

For a comparison of the improvements in PFE, we securely evaluate our generated UCs with the GMW-based SP-LUT protocol [11] and Yao's GC protocol [49]. In Tab. 5, we show the runtime and communication of our LUT-based UC construction (LUC) compared to the most recent UC construction of Liu et al. [32] as baseline. Our new UC construction is the fastest implementation: Compared to the baseline using Yao [49], the total runtime for our sample circuits is faster by a factor of $1.14 - 2\times$. The communication improvements over the baseline using Yao [49] is $1.13 - 2.25\times$. The runtime of Yao's protocol is $3.83 - 11.5\times$ faster than that of the LUT-based protocols which can be explained

Table 4: Comparison of the sizes of our LUT-based UC construction (LUC, cf. §4) and the best previous UC construction of Liu et al. [32] as baseline (in number of AND gates). The smallest size is marked in bold and always achieved by our UCs. The sizes for our UC is the best combinations for $(\rho \rightarrow \omega)$-LUT for $\rho \in \{2, ..., 8\}$ inputs and $\omega \in \{1, \ldots, 8\}$ outputs for the benchmarked circuit.

Circuit	Circuit size (# AND gates)		Improvement (×)	LUT sizes ($\rho \rightarrow \omega$)
	UC of [32]	Our LUC		
AES	1,721,094	**1,519,149**	1.13	$2 \rightarrow 1$
DES	1,275,338	**1,130,037**	1.13	$3 \rightarrow 1$
MD5	286,150	**1,724,221**	1.91	$3 \rightarrow 1$
SHA-1	4,872,501	**2,559,602**	1.90	$3 \rightarrow 1$
SHA-256	10,652,234	**5,351,972**	1.99	$3 \rightarrow 1$
add_32	6,926	**3,907**	1.77	$3 \rightarrow 2$
add_64	17,006	**8,963**	1.90	$3 \rightarrow 2$
comp_32	2,519	**1,278**	1.97	$3 \rightarrow 1$
mult_32x32	347,274	**177,081**	1.96	$3 \rightarrow 3$
karatsuba_32x32	286,933	**156,888**	1.83	$3 \rightarrow 3$
Manhattan_dist	327,203	**150,046**	2.18	$3 \rightarrow 2$
Euclidean_dist	1,852,419	**947,679**	1.95	$3 \rightarrow 3$
fp-add_32	113,620	**90,964**	1.25	$3 \rightarrow 1$
fp-mul_32	293,125	**247,859**	1.18	$3 \rightarrow 1$
fp-exp2_32	2,008,269	**1,548,079**	1.30	$3 \rightarrow 1$
fp-div_32	372,101	**236,300**	1.57	$3 \rightarrow 1$
fp-sqrt_32	176,176	**118,873**	1.48	$3 \rightarrow 1$
fp-comp_32	6,387	**5,628**	1.13	$4 \rightarrow 4$
fp-log_32	1,936,813	**1,499,538**	1.29	$3 \rightarrow 1$

by the constant round complexity of Yao's protocol. From Tab. 5, we can observe that the SP-LUT protocol [11] always has the lowest communication, achieving factor $1.19 - 2.44\times$ less communication than Yao's protocol.

VUC Improvement. Tab. 6 shows that our VUC construction which leaks the fanin of the individual LUTs is up to $2.90\times$ smaller than Liu et al.'s UC when evaluated with Yao's protocol [49], the total runtime for our sample circuits is faster by a factor of $1.1 - 2.85\times$ and the communication is improved by $1.06 - 2.96\times$.

Note that during the process of compiling our VUC construction, our tool conducts an initial verification to determine whether the LUC construction results in a superior size, and, if so, proceeds to compile an LUC. Nonetheless, in the majority of cases, VUC yields a superior size by a factor of up to $1.45\times$. The superiority of VUC over LUC is strongly influenced by the circuit design. Specifically, if the circuit can primarily be constructed using Look-Up Tables (LUTs) with identical input arities, the overall size is superior in comparison

Table 5: Running time and communication for our LUT-based UC construction (cf. §4) compared to the state-of-the-art UC of [32] when evaluated with ABY [10]. We include the LAN evaluation time (in seconds) and the total communication (in Megabytes) between the parties in LUT-based [11] as well as in Yao sharing [49]. The best values are marked in bold.

UC construction	UC of [32]		Our LUT-based UC (LUC)			
MPC protocol	Yao [49]		Yao [49]		SP-LUT [11]	
Circuit	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)
AES	1.926	83.170	**1.608**	69.343	13.187	**28.427**
DES	1.282	57.271	**1.124**	50.570	9.233	**24.311**
MD5	3.471	148.348	**1.832**	76.638	26.642	**46.013**
SHA1	5.184	220.065	**2.756**	113.859	27.268	**58.641**
SHA-256	11.571	481.412	**5.878**	238.364	54.082	**123.045**
add_32	0.018	0.314	**0.009**	0.177	0.224	**0.148**
add_64	0.026	0.770	**0.017**	0.404	0.452	**0.319**
comp_32	0.008	0.117	**0.004**	0.062	0.139	**0.055**
mult_32x32	0.350	15.626	**0.212**	7.300	4.144	**4.531**
karatsuba_32x32	0.292	12.901	**0.191**	6.469	3.685	**4.021**
md256	0.337	14.801	**0.193**	6.592	4.234	**4.544**
ed64	1.924	83.552	**1.046**	39.704	17.524	**24.572**
FP-Add_32	0.164	5.105	**0.139**	4.003	2.903	**2.426**
FP-Mul_32	0.350	13.178	**0.308**	10.949	6.217	**4.579**
FP-Exp2_32	2.292	90.555	**1.612**	68.651	21.531	**38.330**
FP-Div_32	0.458	16.743	**0.296**	10.443	5.918	**6.528**
FPSqrt_32	0.223	7.915	**0.168**	5.237	3.417	**3.442**
FP-comp_32	0.014	0.290	**0.012**	0.235	0.226	**0.118**
FP-log_32	2.083	87.330	**1.600**	66.510	20.198	**36.287**

to VUC. However, if the circuit can be effectively constructed using LUTs with differing input arities, VUC performs better.

Table 6: Sizes, runtime, and communication for our VUC construction (cf. §5). We include the LAN evaluation time (in seconds) and the total communication (in Megabytes) between the parties in LUT-based [11] as well as in Yao sharing [49]. We show the size improvement of VUC over the UC of [32] and LUC construction (cf. §4) in the last two columns.

Circuit	Size	Yao [49]		SP-LUT [11]		Size Improv. (×) VUC/UC of [32]	Size Improv.(×) VUC/LUC
		Time	Comm.	Time	Comm.		
AES	1,519,149	1.61	69.34	13.19	28.43	1.13	1
DES	974,733	0.98	43.44	8.6	22.2	1.31	1.16
MD5	1,191,566	1.22	52.89	23.01	42.77	2.76	1.45
SHA-1	2,559,602	2.76	113.86	27.27	58.64	1.90	1
SHA-256	4,591,982	4.91	201.98	52.22	108.43	2.32	1.17
add_32	3,907	0.01	0.18	0.23	0.15	1.77	1
add_64	8,963	0.02	0.40	0.45	0.32	1.90	1
comp_32	1,188	0.01	0.05	0.04	0.04	2.12	1.08
mult_32x32	130,053	0.14	5.51	1.42	4.06	2.67	1.36
karatsuba_32x32	112,829	0.12	5.01	1.41	3.41	2.54	1.40
Manhattan_dist	112,829	0.13	5.01	1.37	4.09	2.90	1.33
Euclidean_dist	947,679	1.05	39.70	17.52	24.58	1.95	1
fp-add_32	90,964	0.14	4.00	2.90	2.43	1.25	1
fp-mul_32	185,968	0.18	8.11	2.04	3.65	1.58	1.33
fp-exp2_32	1,265,869	1.34	55.72	19.38	25.16	1.59	1.22
fp-div_32	181,904	0.18	7.89	1.91	5.27	2.05	1.30
fp-sqrt_32	89,311	0.10	3.84	1.07	2.70	1.97	1.33
fp-comp_32	5,269	0.01	0.22	0.11	0.093	1.21	1.07
fp-log_32	1,230,530	1.31	54.16	16.17	24.69	1.57	1.22

References

1. Synopsys Inc. design compiler. `http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler` (2010)
2. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. JoC (1990)
3. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and Scalable Universal Circuits. JoC (2020)
4. Attrapadung, N.: Fully Secure and Succinct Attribute Based Encryption for Circuits from Multi-linear Maps (2016)
5. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A., Schneider, T.: Secure Evaluation of Private Linear Branching Programs with Medical Applications. In: ESORICS (2009)
6. Berkeley Logic Synthesis and Verification Group: ABC: A system for sequential synthesis and verification, `http://www.eecs.berkeley.edu/~alanmi/abc/`
7. Bhandari, J., Moosa, A.K.T., Tan, B., Pilato, C., Gore, G., Tang, X., Temple, S., Gaillardon, P., Karri, R.: Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction. CoRR: abs/2111.04222 (2021)
8. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving Remote Diagnostics. In: CCS (2007)
9. Cook, S.A., Hoover, H.J.: A Depth-Universal Circuit. SIAM J. Computing (1985)
10. Demmler, D., Schneider, T., Zohner, M.: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In: NDSS (2015)
11. Dessouky, G., Koushanfar, F., Sadeghi, A., Schneider, T., Zeitouni, S., Zohner, M.: Pushing the Communication Barrier in Secure Computation using Lookup Tables. In: NDSS (2017)
12. Diestel, R.: Graph Theory. Graduate Texts in Mathematics, Springer (2010)
13. Fiore, D., Gennaro, R., Pastro, V.: Efficiently Verifiable Computation on Encrypted Data. In: CCS (2014)
14. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving Credit Checking. In: ACM conference on Electronic Commerce (2005)
15. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-Based Encryption for Circuits from Multilinear Maps. In: CRYPTO (2013)
16. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. In: CRYPTO (2010)
17. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: STOC (1987)
18. Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Poster: Framework for Semi-Private Function Evaluation with Application to Secure Insurance Rate Calculation. In: CCS (2019)
19. Günther, D., Kiss, Á., Schneider, T.: More Efficient Universal Circuit Constructions. In: ASIACRYPT (2017)
20. Henecka, W., Kögl, S., Sadeghi, A., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-Party Computations. In: CCS (2010)
21. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-Complexity Private Function Evaluation is Practical. In: ESORICS (2020)
22. Kamali, H.M., Azar, K.Z., Gaj, K., Homayoun, H., Sasan, A.: LUT-Lock: A Novel LUT-based Logic Obfuscation for FPGA-Bitstream and ASIC-Hardware Protection. In: ISVLSI (2018)
23. Kamara, S., Raykova, M.: Secure Outsourced Computation in a Multi-Tenant Cloud. In: IBM Workshop on Cryptography and Security in Clouds (2011)

24. Karatsuba, A.A., Ofman, Y.P.: Multiplication of Many-Digital Numbers by Automatic Computers. In: SSSR Academy of Sciences (1962)
25. Karnaugh, M.: The Map Method for Synthesis of Combinational Logic Circuits. Transactions of the American Institute of Electrical Engineers (1953)
26. Katz, J., Malka, L.: Constant-Round Private Function Evaluation with Linear Complexity. In: ASIACRYPT (2011)
27. Kiss, Á., Schneider, T.: Valiant's Universal Circuit is Practical. In: EUROCRYPT (2016)
28. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: CANS (2009)
29. Kolesnikov, V., Schneider, T.: A Practical Universal Circuit Construction and Secure Evaluation of Private Functions. In: FC (2008)
30. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: ICALP (2008)
31. Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant's Universal Circuit: Improvements, Implementation, and Applications (2016)
32. Liu, H., Yu, Y., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the Limits of Valiant's Universal Circuits: Simpler, Tighter and More Compact. In: CRYPTO (2021)
33. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - Secure Two-Party Computation System. In: USENIX Security (2004)
34. Masserova, E., Garg, D., Mai, K., Pileggi, L., Goyal, V., Parno, B.: Logic Locking-Connecting Theory and Practice (2022)
35. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In: USENIX Security (2021)
36. Patra, A., Schneider, T., Suresh, A., Yalame, H.: SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation. In: IEEE HOST (2021)
37. Paus, A., Sadeghi, A.R., Schneider, T.: Practical Secure Evaluation of Semi-Private Functions. In: ACNS (2009)
38. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: ASIACRYPT (2009)
39. Pohle, E., Abidin, A., Preneel, B.: Poster: Fast Evaluation of S-boxes in MPC. In: NDSS (2022)
40. Quine, W.V.: The Problem of Simplifying Truth Functions. The American Mathematical Monthly (1952)
41. Sadeghi, A.R., Schneider, T.: Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In: ICISC (2008)
42. Shannon, C.E.: The Synthesis of Two-Terminal Switching Circuits. Bell Syst. Tech. J. (1949)
43. Smart, N., Tillich, S.: Bristol Fashion MPC circuits. `https://homes.esat.kuleuven.be/~nsmart/MPC/old-circuits.html`
44. Valiant, L.G.: Universal Circuits (Preliminary Report). In: STOC (1976)
45. Wegener, I.: The Complexity of Boolean Functions. John Wiley; Sons, Inc. (1987)
46. Wolf, C., Glaser, J., Kepler, J.: Yosys – A Free Verilog Synthesis Suite. In: Austrian Workshop on Microelectronics (2013)
47. Yao, A.C.: How to Generate and Exchange Secrets (Extended Abstract). In: FOCS (1986)
48. Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J., Sinanoglu, O.: Provably-Secure Logic Locking: From Theory To Practice. In: CCS (2017)
49. Zahur, S., Rosulek, M., Evans, D.: Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In: EUROCRYPT (2015)

50. Zhao, S., Yu, Y., Zhang, J., Liu, H.: Valiant's Universal Circuits Revisited: An Overall Improvement and a Lower Bound. In: ASIACRYPT (2019)
51. Zimmerman, J.: How to Obfuscate Programs Directly. In: EUROCRYPT (2015)

A Proof of Proposition 1

Proof. Let $\hat{G} = (\hat{V}, \hat{E}, P)$ be the merging of the $\Gamma_\rho(n)$ EUG $G = (V, E, P)$ and the $\Gamma_{\bar{\rho}}(n)$ EUG $\bar{G} = (\bar{V}, \bar{E}, P)$. Let $G' = (P, E') \in \Gamma_{\rho+\bar{\rho}}(n)$ be the order-preserving graph to be edge-embedded into $\hat{G}$. By Kőnig's theorem [12, Prop. 5.3.1], we can partition E' into disjoint E'_1 and E'_2 such that $E' = E'_1 \cup E'_2$, $G'_1 = (P, E'_1) \in \Gamma_\rho(n)$ and $G'_2 = (P, E'_2) \in \Gamma_{\bar{\rho}}(n)$. Then G'_1 and G'_2 can be edge-embedded into G and $\bar{G}$ respectively. Define $\psi_{\hat{G}} \colon E' \to \mathcal{P}_{\hat{G}}$ as follows:

$$\psi_{\hat{G}}(e') \mapsto \begin{cases} \psi_G(e'), & \text{if } e' \in E'_1, \\ \psi_{\bar{G}}(e'), & \text{if } e' \in E'_2. \end{cases}$$

Since $E' = E'_1 \cup E'_2$, and all edges in E'_1 and E'_2 were edge-embedded into G and $\bar{G}$, each edge is mapped to a path. Furthermore, these paths are edge disjoint because E and $\bar{E}$, in which E'_1 and E'_2 were edge-embedded, are disjoint. $\square$

B UC Compilation

Let C denote the circuit to be embedded and ρ the maximum fan-in of the circuit. We implement UC compilation as follows:

1. Parsing the circuit: The circuit is input in the Secure Hardware Definition Language (SHDL) [33] and parsed into the internal graph representation. If the fan-out of the graph is higher than the allowed fan-in (ρ for the LUC construction cf. §4 and 2 for the VUC construction cf. §5), the fan-out is reduced by copy gates. G denotes the resulting graph. If we want to use multi-input gates in VUC, then the auxiliary graph as in Theorem 3 is generated (cf. Alg. 3). In this case, we denote the auxiliary graph by G and the former graph with possibly reduced fan-out by $\bar{G}$.

2. Splitting G into Γ_1 graphs and creating Γ_1 EUGs: Using the LUC construction yields ρ Γ_1 graphs. For each Γ_1 graph, we create a Γ_1 EUG. Possible EUGs are Valiant's EUG [44] and the 2-way split EUG of Liu et al. [32]. If we use the VUC construction, we get two Γ_1 graphs.

3. Edge-embedding the Γ_1 graphs and merging them: Each Γ_1 graph is edge-embedded into the corresponding Γ_1 EUG. This edge-embedding is coded directly into the control bits of the X- and Y-Switches of the EUG. We do not construct an explicit edge-embedding map ψ because we only need the control bits to create the UC and the programming bits. The concrete algorithm uses a slightly modified version of the edge-embedding algorithm in [19]. Then, the Γ_1 EUGs are merged into a Γ_ρ EUG (for the LUC construction) or into a Γ_2 EUG (for the VUC construction).

4. Doing basic optimizations and checking the correctness of the edge-embedding: We remove edges connecting to an input pole as they will never

be used and replace copy gates with wires. Then we remove isolated nodes or change X- to Y-Switching nodes if one edge was removed before. We check the correctness of the edge-embedding by checking for each edge (u, v) in G, if there is a path leading from u to v.
5. Setting the gates of the EUG: In the VUC construction, we replace the auxiliary poles with wires connecting directly to the actual pole or a Y-Switch if only one input is forwarded. Analogously to step 4, we check the correctness of the edge-embedding to $\bar{G}$. For each node in G, we set the function bits of the corresponding EUG pole. We determine the order of inputs and then set the function bits accordingly. This also involves padding the function bits if the gate has more inputs. Note that these additional inputs are likely to occur since each Universal Gate outputs ρ (in LUC construction) or 2 (in VUC construction) wires, independent of whether they are used in G or not. We pad the function bits such that additional and undesired inputs are ignored.
6. Transforming the EUG into an ABY compatible UC: As a final step, we topologically order the EUG and output it in the UC format compatible with ABY [10]. Then, each node, along with its ongoing and outgoing wires, is written into a circuit file. At the same time, the programming bits are written into a separate programming bits file.