

# Breaking the Size Barrier: Universal Circuits meet Lookup Tables

Yann Disser , Daniel Günther , Thomas Schneider , Maximilian Stillger, Arthur Wigandt, and 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 generalize UCs to simulate circuits consisting of  $(\rho \rightarrow \omega)$ -Lookup Tables (LUTs) that map  $\rho$  input bits to  $\omega$  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  $1.12\times - 2.18\times$  for common functions. Our results show that the greatest size improvement is achieved for  $\rho = 3$  inputs, and it decreases for  $\rho > 3$ .

Furthermore, we introduce Varying Universal Circuits (VUCs), which reduce circuit size at the expense of leaking the number of inputs  $\rho$  and outputs  $\omega$  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  $\mathcal{U}(x, p^C) = C(x)$ .

Valiant [52] proposed the first two UC constructions known as *2-way* and *4-way* split UCs with asymptotically optimal size  $\Theta(n \log n)$  and depth  $\mathcal{O}(n)$ , where  $n = n_i + n_g + n_o$  is the size of the simulated circuit  $C$ . Cook and Hover [13] 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  $\mathcal{O}(d)$ . Kolesnikov and Schneider [35] gave the first practical implementation of a UC of non-optimal asymptotic size  $\mathcal{O}(n \log^2 n)$ . A line of work [32,37,23,7,58] followed with the common goal to minimize the size of Valiant's UC construction. Recently, Liu et al. [38] provided today's most efficient UC construction of size  $\sim 3n \log n$ .

All of these works designed UCs to simulate Boolean gates with 2 inputs and 1 output. However, Valiant's UC construction can be generalized to simulate circuits with  $(\rho \rightarrow 1)$ -LUT, namely *Lookup-Tables* with  $\rho$  inputs  $x_1, \dots, x_\rho$  and one output  $y$  and can compute arbitrary functionalities  $f$  as  $y = f(x_1, \dots, x_\rho)$  [49].

In this work, we propose LUT-based UCs (LUC) that evaluate circuits composed of  $(\rho \rightarrow \omega)$ -LUTs having  $\omega$  output bits  $y_1, \dots, y_\omega$  and are programmed to compute  $y_i = f^i(x_1, \dots, x_\rho)$  for  $1 \leq i \leq \omega$  and an arbitrary functionality  $f^i$ . In addition, we introduce *Varying UCs* (VUCs) that can simulate circuits consisting of  $(\rho \rightarrow \omega)$ -LUTs with varying numbers of inputs  $\rho$  and outputs  $\omega$ , thereby leaking the number of in- and outputs of each LUT. VUCs have various applications (summarized in §1.1) like logic locking [56], which enables the designer to provide the foundry of a chip with a "locked" version of the original circuit. Once the locked circuit on the chip 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 and analyze the size optimization of simulating LUT-based circuits with LUCs and VUCs compared to using traditional Boolean circuit-based UCs.

## 1.1 Applications of (Varying) Universal Circuits

The most prominent application for UCs is **Private Function Evaluation (PFE)** [6], which can be seen as a generalization of *Secure Multi-Party Computation (MPC)* [55,21]. In MPC, a set of  $k$  parties  $\mathcal{P}_1, \dots, \mathcal{P}_k$  jointly compute a publicly known circuit  $C$  on their respective private inputs  $x_1, \dots, x_k$  and obtain nothing but the result  $C(x_1, \dots, x_k)$ . 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  $x_2, \dots, x_k$  run a protocol that yields nothing but  $C(x_2, \dots, x_k)$  and parties  $\mathcal{P}_{2 \leq i \leq k}$  do not learn any information about the circuit  $C$ .

PFE can be implemented via MPC by means of UCs as follows: The parties  $\mathcal{P}_1, \dots, \mathcal{P}_k$  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  $x_2, \dots, x_k$  of parties  $\mathcal{P}_2, \dots, \mathcal{P}_k$ , resulting in  $\mathcal{U}(p^C, x_2, \dots, x_k) = C(x_2, \dots, x_k)$ . 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 *secure outsourcing* [28] and *non-interactive computation* [37]. 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.<sup>1</sup>

As a trade-off between privacy and efficiency, a variant of PFE called **Semi-Private Function Evaluation (SPFE)** was proposed [22,44]. Unlike PFE, SPFE does not hide the entire function, but leaks the topology of certain sub-functions. SPFE can be applied in PFE scenarios where specific function components are known publicly. This approach is particularly useful in cases where certain function details have already been disclosed, often for promotional purposes. An example of this is car insurance companies offering discounts to experienced drivers.

Beyond (S)PFE, UCs have many applications like hiding policy circuits in attribute-based encryption [8,19], multi-hop homomorphic encryption [20], verifiable computation [17], program obfuscation [59], and hardware logic locking [41].

In this work, we introduce **Varying Private Function Evaluation (VPFE)** whose privacy-guarantee lies inbetween PFE and SPFE. Similar to PFE, our Varying UCs (VUCs) for VPFE hide the topology and functionality of the LUTs in the circuit, but leak their number of in- and outputs. This has applications in logic obfuscation techniques called logic locking, as demonstrated in prior studies such as LUT-Lock [27] and eFPGA [11]. 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 [11, 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 [11, Tab. 5]. Therefore, our VUCs can be used for secure logic locking while additionally hiding the topology of the circuit.

## 1.2 Outline and Our Contributions

So far, UC-based PFE research considered synthesis of the input circuit (to generate a small 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 in §2 and summarizing the two UC constructions of Valiant [52] (§3.2) and Liu et al. [38] (§3.3), we contribute the following:

**LUT-based UC (LUC) Construction (§4).** Valiant’s UC construction can be generalized to support the evaluation of  $(\rho \rightarrow 1)$ -LUT-based circuits by merging for the  $\rho$  inputs  $\rho$  instances of its basic building block called edge-universal graph [49, App. A]. This leads to a total size of  $\sim 1.5\rho n \log n$  using Liu et al.’s [38] UC construction. In our work, we extend this into a novel UC construction to simulate for the first time functions composed of  $(\rho \rightarrow \omega)$ -LUTs with  $\rho$  inputs and  $\omega$  outputs. Our construction is general, can be applied to all UC constructions based on Valiant’s framework [52] and improvements by Liu et al. [38], and fits into the definition of UCs (cf. Def. 1 on page 4).

**Size Improvements of LUCs for Basic Primitives (§4.3).** Tab. 1 shows the history of improvements in UC sizes. Taking (V)PFE as our greatest motivation for improving UC sizes, we study three basic building blocks that can be used to construct more complex functionalities for common PFE applications:

<sup>1</sup> UC-based PFE, unlike PFE based on Fully Homomorphic Encryption (FHE) [20,33], relies primarily on symmetric encryption and involves far less computation.

Table 1: Asymptotic sizes of various UC constructions and improvements over previous works. Previous UCs were for 2-input gates, whereas our LUC construction is generalized to  $\rho$ -input LUTs.

Universal Circuit	Asymptotic Size	Improvement over previous work	Fanin
Valiant’s 2-way [52]	$5n \log n$	-	2
Valiant’s 4-way [52]	$4.75n \log n$	$1.05\times$	2
Zhao et al.’s 4-way [58]	$4.5n \log n$	$1.06\times$	2
Liu et al.’s 2-way [38]	$3n \log n$	$1.5\times$	2
<b>Our LUC</b>	<b><math>1.5\rho n \log n</math></b>	<b><math>1.12 - 2.18\times</math></b>	<b><math>\rho</math></b>

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  $1.67\times$  for full adders,  $2.67\times$  for comparators, and  $2\times$  for multiplexers.

**Varying UC (VUC) Construction (§5).** In several applications (cf. §1.1, §5.2) only the programming of the LUTs needs to be hidden, but not their dimensions, i.e., we can leak their number of in- and outputs. 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 LUTs’ number of in- and outputs. We give the first VUC construction that eliminates the leading  $\rho$  factor of our LUC construction (cf. Tab. 1), 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. [38] which is of independent interest and our LUC and VUC constructions.<sup>2</sup> Moreover, we integrate these three UC implementations into the MPC framework ABY [14] for PFE. To create LUT-based circuits, we used the hardware circuit synthesis tool Yosys-ABC [54,10] and Synopsis Design Compiler [5] for LUT-Mapping. We optimize LUT-based PFE by combining LUTs with overlapping inputs and multiple outputs. However, hardware synthesis tools do not by default support mapping to multiple output LUTs. To address this, we post-process the single-output LUT circuits produced by the synthesis tool to convert them to multi-output LUT circuits.

**Evaluation (§6.3).** We experimentally evaluate our LUC and VUC constructions for various LUT sizes, and compare them with the previous best construction of Liu et al. [38]. 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 [38] in terms of circuit sizes by up to  $2.18\times$ .

### 1.3 Related Work

**Universal Circuits (UCs).** Valiant [52] 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 be generalized to simulate circuits composed of  $(\rho \rightarrow 1)$ -LUTs as shown by Sadeghi and Schneider [49, App. A] which is summarized in §4.1.

A modular UC construction of non-optimal size  $\sim 1.5n \log^2 n + 2.5n \log n$  was proposed and implemented by Kolesnikov and Schneider [35]. Their construction beats Valiant’s construction for small circuits thanks to small prefactors. Motivated to provide more efficient PFE, Kiss and Schneider [32] implemented Valiant’s 2-way split construction and proposed a more efficient hybrid construction combining the 2-way split construction with the modular construction of [35]. Lipmaa et al. [37] 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. [23] modularized Valiant’s construction, implemented the more efficient 4-way split construction, gave a generic edge-embedding algorithm for  $k$ -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. [58] 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. [7] proposed and implemented a scalable hybrid UC construction combining Valiant’s 2-way and 4-way split constructions

<sup>2</sup> Our code is published under the MIT license at: <https://crypto.de/code/LUC>.

with Zhao et al.’s improvements [58]. Most recently, Liu et al. [38] 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 split 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 [31] designed a constant-round two-party PFE protocol with linear communication complexity based on homomorphic public-key encryption. Holz et al. [25] optimized and implemented the protocol of [31], demonstrating its superiority over the hybrid UC implementation of Alhassan et al. [7] already for circuits with a few thousand gates. Liu et al. [39] provide a constant-round actively secure two-party PFE protocol with linear complexity. However, all these protocols are not generic and hence not directly compatible with arbitrary MPC frameworks, which makes them less flexible. For instance, these protocols cannot easily be extended to multiple parties. Ji et al. [26] demonstrated the evaluation of private RAM programs using four servers, building the first PFE of non-Boolean and non-arithmetic functions. 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 and implementation was proposed by Paus et al. [44] who provided several 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. [22] built 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 experienced drivers usually get discounts.

**MPC on LUTs.** In the area of secure multi-party computation (MPC), prior work noticed that 2-input/1-output gates can be extended into multi-input/multi-output gates to reduce the circuit evaluation overhead [40,24,46,15,42]. In Yao’s Garbled Circuit (GC) setting, Fairplay [40] implemented MPC protocols to evaluate gates with up to 3-input gates. The TASTY framework [24] implemented  $\rho$ -input garbled gates using the garbled row reduction optimization [45]. Recently, [46] 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 in the secret-sharing setting aims to optimize the rounds and communication of the online phase without using Yao’s GC protocol: [15] extended 2-input AND gates to the general  $N$ -input case using LUTs. Recently, ABY2.0 [42] 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 [43] handles the circuit generation with multi-input gates by using industry-grade hardware synthesis tools [54,10].

## 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  $n_o$ :  $n = n_i + n_g + n_o$ .

**Definition 1 (Universal Circuit [52,7]).** *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 programming bits  $p^C$  such that  $\mathcal{U}(x, p^C) = C(x)$  for any input  $x \in \{0, 1\}^{n_i}$ .*

### 2.1 Graph Theory

Let  $G = (V, E)$  be a directed graph and  $v \in V$ . The indegree (resp. outdegree) of  $v$  which is the number of incoming (resp. outgoing) edges is denoted by  $\deg^+(v)$  (resp.  $\deg^-(v)$ ).  $G$  has fanin (resp. fanout)  $\rho$  if  $\deg^+(v) \leq \rho$  (resp.  $\deg^-(v) \leq \rho$ ) for all  $v \in V$ . We denote by  $\Gamma_\rho(n)$  all directed acyclic graphs with at most  $n$  nodes and fanin/fanout  $\rho$  for  $\rho, n \in \mathbb{N}$ . For  $U \subset V$ ,  $G[U] := \{U, \{e = (u, v) \in E : u, v \in U\}\}$  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  $\eta_G: V \rightarrow \{1, \dots, |V|\}$  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 [52,32,37,23,58,38] 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 [52] and Liu et al.'s [38] constructions. On a high level, a universal circuit (UC) for simulating circuits  $C \in \Gamma_\rho(n)$  is composed of  $\rho$  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  $\rho$  of the gates in the simulated circuit  $C$ .

**Definition 2 (Edge-Embedding [52,37,7,38]).** Let  $G = (V, E)$  and  $G' = (P, E')$  be directed graphs with  $P \subset V$  and  $G'$  acyclic. An edge-embedding from  $G'$  into  $G$  is a map  $\psi: E' \rightarrow \mathcal{P}_G$ , 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 [52,37,7,38]).** A directed graph  $G = (V, E)$ , denoted as  $\mathcal{U}_\rho(n)$  with ordered pole set  $P := \{p_1, \dots, p_n\} \subset V$  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  $G = (V, E)$ , 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) [52,32,37,23,58,7,38]. 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  $\rho$  instances of  $\Gamma_1(n)$  EUGs (cf. Cor. 1).

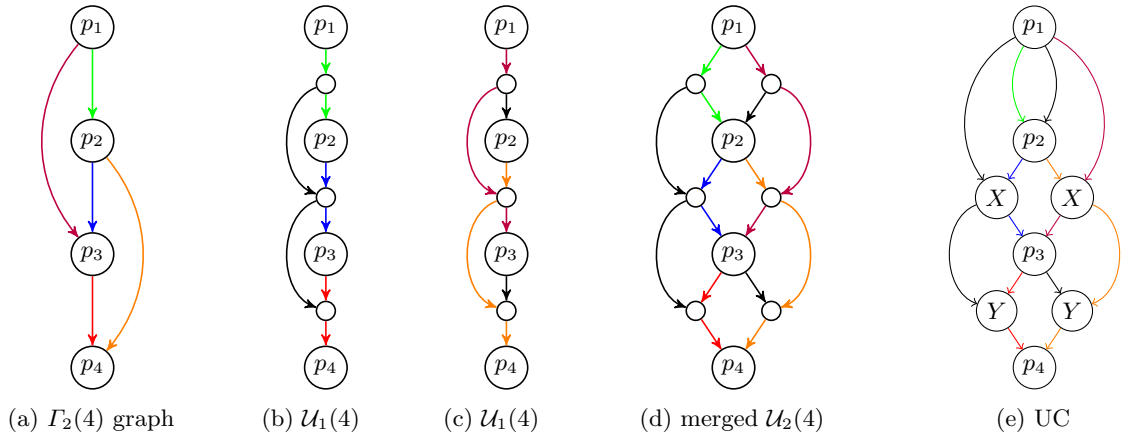


Fig. 1: (a) shows the  $\Gamma_2(4)$  graph with already partitioned edge sets  $E_1$  and  $E_2$ , (b) and (c) show the EUGs in which the edge sets  $E_1$  resp.  $E_2$  are embedded, (d) shows the merged EUG with all edges embedded, (e) shows the resulting UC, where  $p_1$  is an input, and  $p_2,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

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

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

We prove Prop. 1 in App. A.

**Corollary 1 ([52, Corollary 2.2]).** An EUG for  $\Gamma_\rho(n)$  can be constructed by merging  $\rho$  EUGs for  $\Gamma_1(n)$ .

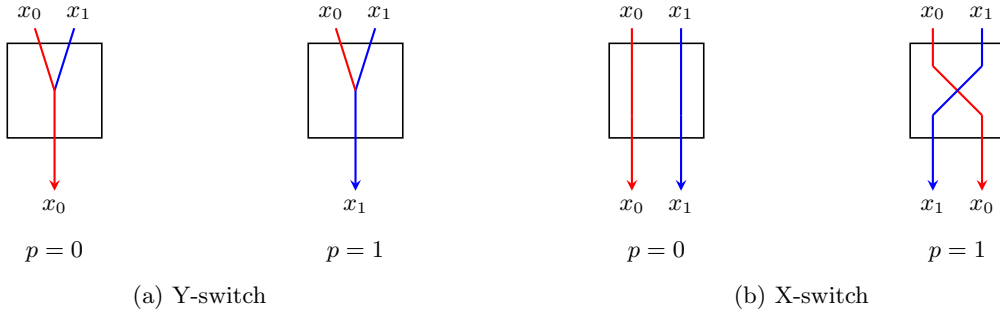


Fig. 2: Switching blocks with programming bit  $p$  (from [35]).

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

We call the UCs that are constructed according to Cor. 1 *LUT-based UCs* (LUCs) and this construction was first mentioned in [49, App. A]. 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  $\rho$ .

## 2.2 Building Universal Circuits from Edge-Universal Graphs

**Boolean Circuits.** A Boolean circuit is a directed acyclic graph whose nodes are Boolean inputs, (binary) gates, and outputs, with directed edges representing the wires. A Boolean gate is a function  $z: \{0, 1\}^k \rightarrow \{0, 1\}$  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 [50]. Unfortunately, we cannot avoid an exponential blow-up of the number of gates by this transformation [53, Theorem 2.1]. The two most prominent minimization methods for Boolean circuits are due to Karnaugh [30] and Quine-McCluskey [47]. As already mentioned, the UC constructions by Valiant [52] and Liu et al. [38] are designed to embed  $\Gamma_\rho(n)$  graphs, thus we possibly need to reduce the outdegree of the gates to  $\rho$  by using so-called copy gates which just copy their inputs [52, Corollary 3.1].<sup>3</sup>

**From Edge-Universal Graphs to Universal Circuits.** The translation from an EUG  $G = (V, E)$  into a UC is depicted in Fig. 1 and works as follows. First, the nodes of circuit  $C$  to be embedded in  $G$  are considered as the poles  $P \subset V$  of the EUG. A pole  $p \in P$  is translated into an input or output wire, if  $p$  corresponds to an input or 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 [52,7,38]),  $2^k$  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 (Y-switches) spanned over the  $2^k$  programming bits, where the correct programming bit specified by the  $k$  inputs is forwarded to the output (more details in [7,38]).<sup>4</sup>

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 [35].
- 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 [35].
- 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.

<sup>3</sup> 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.

<sup>4</sup> In Yao’s garbled circuit protocol [55], the UC’s universal gates can be implemented as garbled tables 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 [52] (§3.2), and describe the state-of-the-art construction of Liu et al. [38] (§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  $\rho$  instances of these (cf. Cor. 1) to construct a  $\Gamma_\rho(n)$  EUG ( $\rho = 2$  for binary gates), and translating this EUG into a UC. Valiant [52] proposed the first two designs for  $\Gamma_1(n)$  EUGs, today known as 2-way and 4-way constructions, having asymptotic sizes of  $\sim 2.5n \log n$  and  $\sim 2.375n \log n$ .<sup>5</sup> Recently, Liu et al. [38] 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  $G = (V, E)$  for a circuit of size  $n$  with a  $k$ -way construction and pole set  $P \subset V$ . 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  $G$ .<sup>6</sup>

More formally, a superpole shall be able to edge-embed any so-called *augmented  $k$ -way block* (similar to an augmented DAG in [38]). 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.  $(p, out) \in E^{io}$ ).

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  $SP(k)$  is a tuple  $SP(k) = (G = (V, E), P, \mathcal{P}, \mathcal{I}, \mathcal{O})$  with pole set  $P \subset V$ , with following conditions:*

- $P = \mathcal{P} \cup \mathcal{I} \cup \mathcal{O}$  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  $\{in_1, in_2, \dots, in_k\}$  and the output recursion points  $\mathcal{O}$  as  $\{out_1, out_2, \dots, out_k\}$ . 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 [52] 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.

#### 3.2 Valiant’s EUG construction [52]

**Definition 7 (Valiant EUG).** *A Valiant EUG  $G = (V, E)$  with pole set  $P \subset V$  and sub-graphs  $G^*, G^1, \dots, G^k$  is created by Alg. 1 (Valiant). We also use the notation  $\text{Valiant}_k(n)$  for a Valiant EUG with  $n$  poles and split parameter  $k$ .*

---

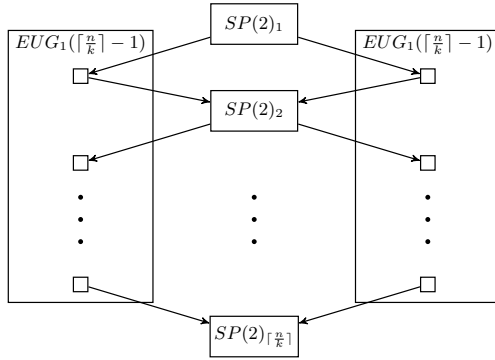
**Algorithm 1: Valiant( $P, k$ )**


---

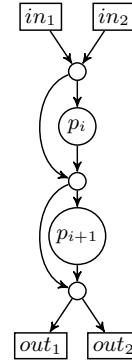
**Input** : Poles  $P := \{p_1, \dots, p_n\}$ , split parameter  $k$   
**Output** :  $\Gamma_1(n)$  EUG  $G = (V, E)$ , pole set  $P$ , sub-graphs  $G^*, G^1, \dots, G^k$

- 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    $\mathcal{P}_i \leftarrow \{p_{k(i-1)+1}, \dots, p_{ki}\}$   
    // Use  $\mathcal{O}_{i-1}$  as input recursion points to this superpole (cf. Fig. 3b)
- 5    $SP(k)_i = (G_i = (V_i, E_i), P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i) \leftarrow \text{Createsuperpole}(\mathcal{P}_i, \mathcal{O}_{i-1}, k); G^* \leftarrow G^* \cup \{G_i\}$
- 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      $G^i \leftarrow (\emptyset, \dots, \emptyset)$  // Recursion base
- 10  else
- 11     // Take the  $i$ -th output recursion point of each superpole (but the last) as the poles for the next sub EUG
- 12      $P^i \leftarrow \{\mathcal{O}_1[i], \mathcal{O}_2[i], \dots, \mathcal{O}_{\lceil \frac{n}{k} \rceil - 1}[i]\}$
- 13      $(G^i = (V^i, E^i), \dots) \leftarrow \text{Valiant}(P^i, k)$
- 13      $V \leftarrow V \cup V^i, E \leftarrow E \cup E^i$
- 14 return  $G = (V, E), P, G^*, G^1, \dots, G^k$

---



(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.



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  $SP(k)_1 = (G_1 = (V_1, E_1), P_1, \mathcal{P}_1, \mathcal{I}_1, \mathcal{O}_1), \dots, SP(k)_{\lceil n/k \rceil} = (G_{\lceil n/k \rceil} = (V_{\lceil n/k \rceil}, E_{\lceil n/k \rceil}), P_{\lceil n/k \rceil}, \mathcal{P}_{\lceil n/k \rceil}, \mathcal{I}_{\lceil n/k \rceil}, \mathcal{O}_{\lceil n/k \rceil})$  (lines 3-6 in Alg. 1). `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  $\mathcal{O}_1, \dots, \mathcal{O}_{\lceil n/k-1 \rceil}$ , each of size  $k$ , then recursively build the poles of the nested EUGs in the next recursion step (lines 7-13 in Alg. 1), i.e., we build  $k$  nested EUGs  $G^1 = (V^1, E^1), \dots, G^k = (V^k, E^k)$  with pole sets  $P^1, \dots, P^k$ , where  $G^i \in \Gamma_1(\lceil n/k \rceil - 1)$  and  $P^i = (\mathcal{O}_1[i], \dots, \mathcal{O}_{\lceil n/k-1 \rceil}[i])$ . Note that  $\mathcal{I}_i := \mathcal{O}_{i-1}$  for all  $1 < i \leq \lceil n/k \rceil$  as the  $k$  outputs of  $G_i \in SP(k)_i$  are pairwise merged with the respective  $k$  inputs of  $G_{i+1} \in SP(k)_{i+1}$ . 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, we 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 [23,7] 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. We define the following theorem and refer to [7,38] for its proof:

**Theorem 1.** *Let  $G = (V, E)$  be a Valiant EUG with pole set  $P \subset V$  of size  $|P| = n$  and sub-graphs  $G^*, G^1, \dots, G^k$ . Then  $G$  is an EUG for  $\Gamma_1(n)$ .*

### 3.3 Liu et al.’s EUG construction [38]

**Definition 8 (Liu<sup>+</sup> EUG).** *A Liu<sup>+</sup> EUG  $G = (V, E)$  with pole set  $P \subset V$  and sub-graphs  $G^*, G^1, \dots, G^k$  is created by Alg. 2 (Liu<sup>+</sup>). We also use the notation  $\text{Liu}_k^+(n)$  for a Liu<sup>+</sup> EUG with  $n$  poles and split parameter  $k$ .*

We refer to App. B for a complete description of the construction of Liu et al. [38] including Alg. 2. In the subsequent sections of this work, we leverage the following theorem and refer to [38] for its proof:

**Theorem 2 (cf. [38, Theorem 4]).** *Let  $G = (V, E)$  be a Liu<sup>+</sup> EUG with pole set  $P \subset V$  of size  $|P| = n$  and sub-graphs  $G^*, G^1, \dots, G^k$ . Then  $G$  is an EUG for  $\Gamma_1(n)$  with size bounded by*

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

## 4 Evaluating LUTs with UCs

In this section, we extend the UC constructions from §3 to be able to simulate  $(\rho \rightarrow \omega)$ -LUT-based circuits. In §4.1, we first review the construction of [52,35] to evaluate  $(\rho \rightarrow 1)$ -LUT-based circuits, i.e., circuits that consist of LUTs with  $\rho$  inputs and one output. Then, in §4.2, we extend this to our LUT-based UCs (LUCs) that allows the UC to simulate  $(\rho \rightarrow \omega)$ -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.

<sup>5</sup>  $\sim 2.25n \log n$  with the optimizations by Zhao et al. [58] for the 4-way construction.

<sup>6</sup> We distinguish between EUGs and nested EUGs as the recursively constructed nested EUGs differ from its first EUG in Liu et al.’s construction [38].

---

**Algorithm 2:**  $\text{Liu}^+(P, k)$ 

---

**Input** : Poles  $P := \{p_1, \dots, p_n\}$ , split parameter  $k$   
**Output** :  $\Gamma_1(n)$  EUG  $G = (V, E)$ , pole set  $P$ , sub-graphs  $G^*, G^1, \dots, G^k$

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

---

#### 4.1 UCs for LUTs with multiple inputs [52,49]

Valiant [52] proposed a method to integrate LUTs with more than two inputs into UCs and its size has been computed in [49].

We can get a UC with  $n$  copies of  $(\rho \rightarrow 1)$ -LUT from a  $\Gamma_\rho(n)$  EUG that is merged by  $\rho$  instances of  $\Gamma_1(n)$  EUGs according to Cor. 1. Each pole of  $U$  that is not an input or an output can then be implemented as a LUT with  $\rho$  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  $\rho$  instances of  $\text{Liu}_2^+(n)$  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  $(\rho \rightarrow \omega)$ -LUTs with  $\omega > 1$  outputs in UCs, we propose a general solution that is compatible with the original UC constructions of Valiant [52] and Liu et al. [38]. The high level idea is as follows: For every  $(\rho \rightarrow \omega)$ -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

$\rho$  inputs of the LUT using any of the above UC constructions and computes the first output of the LUT. The remaining poles copy the  $\rho$  inputs of the first poles by direct connections and compute the remaining outputs of the LUT, resulting in a chain of  $\omega$  poles.

We define the class of  $\Gamma_{\rho,\omega}(n)$  graphs that is used to map  $n$  ( $\rho \rightarrow \omega$ )-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 (denoted as pole  $v_{i,1}$  in  $G$ ) in total  $\omega - 1$  additional poles (denoted as  $v_{i,2}, \dots, v_{i,\omega}$ ). These added poles  $v_{i,j>1}$  use the inputs from pole  $v_{i,1}$  and thus, they all have in-degree 0 (cf. condition 3 in Def. 9). 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:*

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

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 conditions 3 and 4 in Def. 9. Thus,  $G \in \Gamma_{\rho}(n\omega)$ .  $\square$

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  $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}})$  with a topologically ordered pole set  $P \subset V^{\mathcal{U}}$  that has the form  $(\dots, v_{i-1,\omega}, v_{i,1}, \dots, v_{i,\omega}, v_{i+1,1}, \dots)$  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<sup>+</sup> EUG  $\mathcal{U}$  with pole set  $P$  and split parameter  $k = 2$ . Then we merge  $\rho$  instances of it. By Thm. 2 with  $|SP(2)| = 5$  [38] 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  $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$  with pole set  $P \subset V^{\bar{\mathcal{U}}}$ : 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  $\rho$  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  $P$ .

**Step 3:** Embed any graph  $G = (P, E) \in \Gamma_{\rho,\omega}(n)$  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  $\psi$  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 end up at pole  $v_{i,j>1} \in \bar{\mathcal{U}}$ . This, however, is fulfilled due to condition 3 of Def. 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  $\psi$  directly from Cor. 2.  $\square$

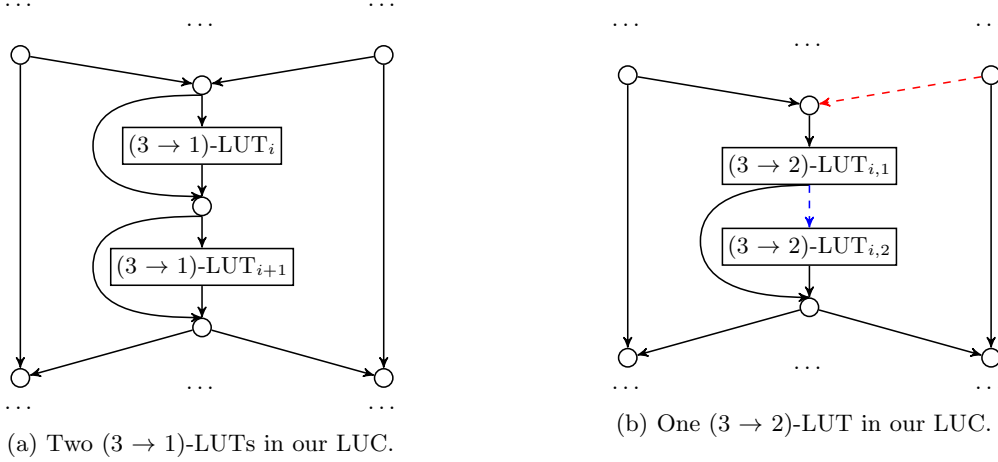


Fig. 4: Embedding of  $(3 \rightarrow 1)$ -LUTs (a) and  $(3 \rightarrow 2)$ -LUTs (b) in a single superpole of our LUC construction. The blue line in (b) indicates that the inputs of the first LUT part are forwarded to the second LUT part. Each of the LUT parts in (b) generate one output with the same inputs, thus building together a  $(3 \rightarrow 2)$ -LUT. The red edge is optional and can be removed as only one input in the superpole is needed.

In Fig. 4, we present our LUC construction for the embedding of both  $(3 \rightarrow 1)$ -LUTs and  $(3 \rightarrow 2)$ -LUTs within a single superpole. Specifically, in Fig. 4a, a superpole consists of two  $(3 \rightarrow 1)$ -LUTs, each having three individual inputs and one output. In contrast, in Fig. 4b, a  $(3 \rightarrow 2)$ -LUT requires two poles, limiting the embedding capacity to a single  $(3 \rightarrow 2)$ -LUT within one superpole. We achieve this by implementing each pole as a  $(3 \rightarrow 1)$ -LUT in our LUC construction, effectively combining them to form a  $(3 \rightarrow 2)$ -LUT. The second part of the LUT shares the same inputs as the first part (indicated by the blue edge in Fig. 4b), eliminating the need for an additional node between the two poles. The two outputs of the  $(3 \rightarrow 2)$ -LUT are forwarded to the lower node and can then propagate to the nested EUGs through this node. As an optimization, we can remove one incoming edge from the superpole (indicated by the red edge in Fig. 4b) since only one outer input is utilized.

### 4.3 Improvement

In this section, we show improvements of our LUC for several basic building blocks like full adder (FA), comparator (CMP), and multiplexer (MUX). As summarized in Tab. 2, our basic building blocks are smaller than the previous constructions [23,32,38,7] in UC size by factor  $\approx 1.67\times - 2.67\times$ . Note that we compute improvement factors based only on the prefactor. The actual enhancements will be greater as also the logarithmic term is improved. This UC size reduction is achieved by merging 2-input gates into larger multi-input LUTs.

*Full Adder (FA):* The optimized implementation of a FA uses four 2-input XOR gates and one 2-input AND gate (cf. [34, Fig. 2]). We can implement a FA using only one  $(3 \rightarrow 2)$ -LUT, resulting in an improvement by  $\approx 1.67\times$  in LUC size (cf. Tab. 2). The embedding of a  $(3 \rightarrow 2)$ -LUT in our LUC is depicted in Fig. 4b.

*Comparator (CMP):* The 1-bit comparator consists of three 2-input XOR gates and one 2-input AND gate (cf. [34, Fig. 6]). Our improved LUT-based instantiating for CMP uses only one  $(3 \rightarrow 1)$ -LUT, resulting in an improvement of  $\approx 2.67\times$  in LUC size (cf. Tab. 2). The embedding of a  $(3 \rightarrow 1)$ -LUT in our LUC is depicted in Fig. 4a.

*Multiplexer (MUX):* The MUX block can be instantiated with two 2-input XOR gates and one 2-input AND gate (cf. [36, Fig. 2]). In our approach, MUX can be instantiated with only one  $(3 \rightarrow 1)$ -LUT, resulting in an improvement of  $\approx 2\times$  in the LUC size (cf. Tab. 2). The embedding of a  $(3 \rightarrow 1)$ -LUT in our LUC is depicted in Fig. 4a.

Table 2: LUC sizes for basic building blocks which can be used to construct more complex functionalities.  $b$  denotes the frequency of occurrence of the specific building blocks within the circuit.

Building Block (BB)	Boolean Circuit		LUT-based Circuit		Improvement
	# Gates	Asympt. UC Size	LUT type	Asympt. LUC Size	
FA	4 XOR 1 AND	$15b \log_2 5b + \mathcal{O}(b)$	(3 → 2)-LUT	$9b \log_2 b + \mathcal{O}(b)$	1.67×
CMP	3 XOR 1 AND	$12b \log_2 4b + \mathcal{O}(b)$	(3 → 1)-LUT	$4.5b \log_2 b + \mathcal{O}(b)$	2.67×
MUX	2 XOR 1 AND	$9b \log_2 3b + \mathcal{O}(b)$	(3 → 1)-LUT	$4.5b \log_2 b + \mathcal{O}(b)$	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. [34, 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. [34, Fig. 3]). Using our FA construction, the LUC size of the addition and subtraction is improved by  $\approx 1.67\times$ .

*Multiplication.* Multiplication of two  $l$ -bit numbers can be composed of  $l^2$  of 2-input AND gates ((2 → 1)-LUT) and  $(l - 1)$   $l$ -bit adders [34]. Using the efficient implementation for LUT-based adders, the LUC size of the multiplication circuit is improved by  $\approx 1.67\times$ .

*Multiplexer.* An  $l$ -bit multiplexer circuit can be composed of  $l$  parallel MUX blocks (cf. [36, Fig. 9]) to select one of the  $l$ -bit inputs. So, using our LUT-based MUX has  $\approx 2\times$  improvement for an  $l$ -bit multiplexer.

*Comparison.* An  $l$ -bit comparison circuit can be composed of a chain of  $l$  CMP blocks (cf. [34, Fig. 5]). Thus, our CMP construction improves the LUC size of the comparison circuit by  $\approx 2.67\times$ . 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. [34, Fig. 8]) and hence is improved by  $\approx 2.3\times$ .

## 5 Our Varying UC (VUC) Construction

In many applications, sub-functionalities are naturally implemented by LUTs with higher dimension, e.g., Sboxes in AES. In this case, we aim to put single LUTs with a higher dimension (e.g.,  $\rho = 8$ ) into the UC. Using our LUC construction 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 → 1)-LUT few times in the whole circuit.<sup>7</sup> Thus, our aim is to find a way to use single LUTs with input dimension of  $\rho > 3$  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 [52,35,32,37,23,7,38]. 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, ( $\rho \rightarrow \omega$ )-LUTs with a globally fixed number of inputs  $\rho$  and outputs  $\omega$ . 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 ( $\rho \rightarrow \omega$ )-LUTs with varying numbers of inputs and outputs, thereby revealing the dimensions of the individual LUTs.

**Definition 10 (Varying Universal Circuit (VUC)).** *A Varying Universal Circuit  $\mathcal{V}$  for  $n_i$  inputs, the ordered list of  $n_g$  gates  $\mathcal{G} = (\mathcal{G}_1, \dots, \mathcal{G}_{n_g})$  of varying input and output dimensions, and  $n_o$  outputs is a Boolean circuit that can be programmed to compute any Boolean circuit  $C$  with  $n_i$  inputs,  $n_o$  outputs, and  $n_g$  gates that can be topologically ordered into  $\mathcal{G}$  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.

<sup>7</sup> An alternative would be to decompose the larger LUTs into multiple smaller ones using Shannon expansion [50].

---

**Algorithm 3:** AuxiliaryGraph( $G$ )

---

**Input** :  $G = (V, E) \in \Gamma_{P^+, 2}(n)$   
**Output** :  $\bar{G} = (\bar{V}, \bar{E}) \in \Gamma_2(n + \Delta)$  with  $\Delta = \sum_{i=0}^n \max\{\lceil \frac{P_i^+ - 2}{2} \rceil, 0\}$

- 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              $j \leftarrow j + 1$

---

### 5.1 The VUC construction

First, we show how to build our VUC for evaluating different  $(\rho \rightarrow 1)$ -LUTs with varying input dimensions  $\rho$ . Later in this section, we show how to extend this construction to evaluate any  $(\rho \rightarrow \omega)$ -LUTs with varying input and output dimensions  $\rho$  and  $\omega$ . 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  $\rho$  instances of the large  $\Gamma_1(n)$  EUG for  $(\rho \rightarrow 1)$ -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 to a real pole  $v$  to push the indegree of  $v$  to  $\rho$ . Def. 11 defines  $\Gamma_{P^+, P^-}(n)$  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  $\rho$  input LUTs by setting  $P^+ = \mathbb{1}_\rho$ ,<sup>8</sup> i.e., each LUT in the circuit can have at most  $\rho$  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** ( $\Gamma_{P^+, P^-}(n)$ ). *Let  $G = (V, E)$  be a directed acyclic graph with topologically ordered  $V := \{v_1, \dots, v_n\}$  and  $P^+, P^- \in \mathbb{N}^n$ . Then  $G \in \Gamma_{P^+, P^-}(n)$  if:*

- $|V| \leq n$ ,
- $\deg^+(v_i) \leq P_i^+ \wedge \deg^-(v_i) \leq P_i^- \quad \forall i \in [n]$ .

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

In this sense, Cor. 2 yields a  $\Gamma_{\rho, \rho}(n)$  EUG. In the following, we describe our VUC construction. An example of the whole EUG creation and the 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  $\Gamma_1(n)$  EUGs, each of the  $n$  poles (excluding inputs and outputs) can take two inputs, and *can*, but not necessarily need to, compute one output. 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  $(5 \rightarrow 1)$ -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 does not compute any output. Instead, the ingoing edges are forwarded to pole  $p_8$  (dashed lines) and the outgoing edges (dotted gray lines) are removed. Pole  $p_8$  now has, in addition to its two regular ingoing edges, three additional 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  $(\rho \rightarrow 1)$ -LUT,

---

<sup>8</sup>  $\mathbb{1}$  denotes the vector where each entry is 1.

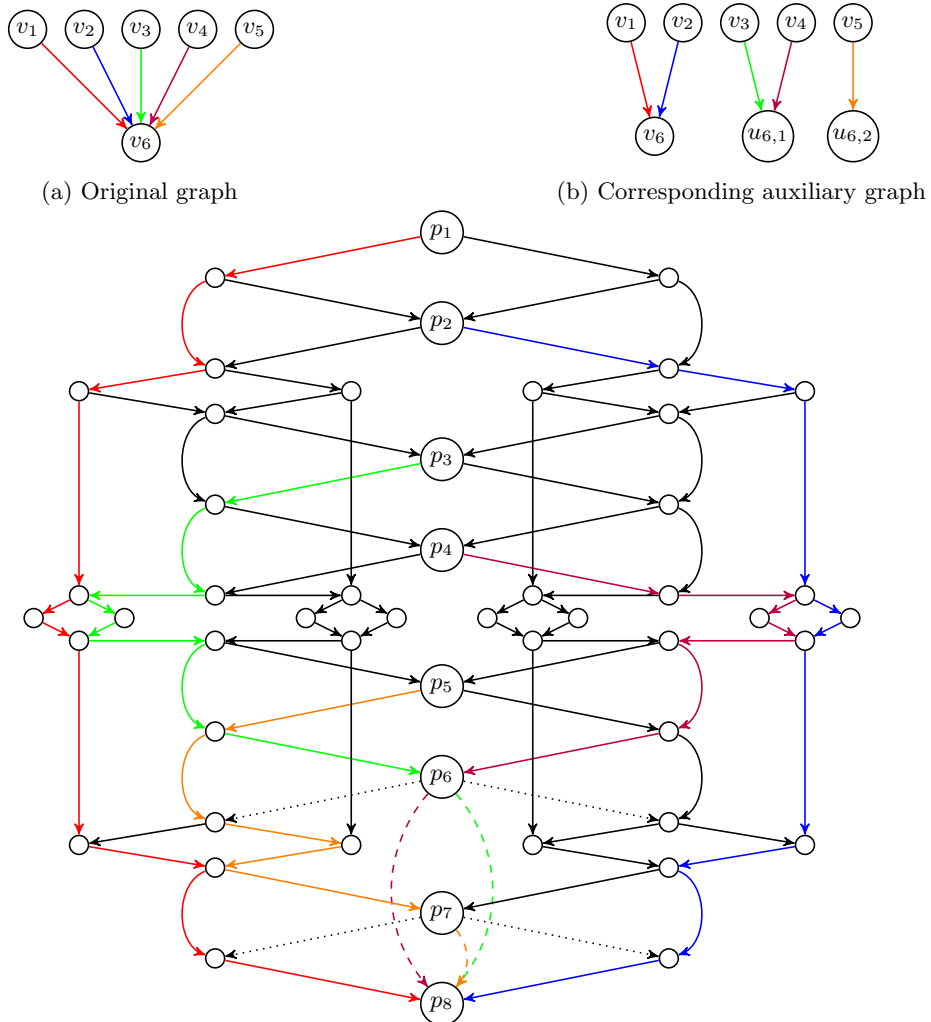


Fig. 5: Our varying UC construction for  $\rho = 5$  inputs.

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

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

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

*Proof. Step 1:* Create a  $\Gamma_2(n + \Delta)$  EUG  $\mathcal{U} = (V^{\mathcal{U}}, E^{\mathcal{U}})$  with a topologically ordered pole set  $P$  that has the form  $(\dots, v_{i-1}, u_{i,1}, \dots, u_{i, \lceil \frac{P_i^+ - 2}{2} \rceil}, v_i, \dots)$  for all  $i \in [n]$ , i.e., the auxiliary poles  $u_{i,j}$  for  $j \in [\lceil \frac{P_i^+ - 2}{2} \rceil]$  are directly preceding the original pole  $v_i$ : We do this by creating a Liu<sup>+</sup> 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  $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$  with pole set  $P \subset V^{\bar{\mathcal{U}}}$ : Let  $u_{i,j}$  be an auxiliary pole of  $v_i$  for  $i \in [n], j \in [\lceil \frac{P_i^+ - 2}{2} \rceil]$ . 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  $G = (P, E) \in \Gamma_{P^+,2}(n)$  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  $\bar{G} = (\bar{V}, \bar{E}) = \text{auxiliaryGraph}(G) \in \Gamma_2(n + \Delta)$  (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  $\psi: \bar{E} \rightarrow \mathcal{P}_{\bar{\mathcal{U}}}$ . To show that  $\bar{\mathcal{U}}$  is a  $\Gamma_{P^+,2}(n)$  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,  $\psi$  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  $\psi$  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  $\bar{\mathcal{U}}$ .  $\square$

Thm. 3 gives us an EUG that can be used to build VUCs for  $(\rho \rightarrow 1)$ -LUTs with varying parameter  $\rho$  and can thus be used for VPFE. Next, we consider VUCs for a fixed constant  $\rho$  which yields classical PFE.

**Corollary 4.** *Let  $P^+ = \mathbb{1}\rho \in \mathbb{N}^n$  for  $\rho > 2$ . Then there exists a EUG for  $\Gamma_{P^+,2}(n)$  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.* We follow the proof of Thm. 3 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  $(\dots, v_{i-1}, u_{i,1}, \dots, u_{i, \lceil \frac{\rho-2}{2} \rceil}, v_i, u_{i+1,1}, \dots, u_{i+1, \lceil \frac{\rho-2}{2} \rceil}, v_{i+1}, \dots)$  as described in step 1 in the proof of Thm. 3.

**Step 2:** Adjust  $\mathcal{U}$  to get the final EUG  $\bar{\mathcal{U}} = (V^{\bar{\mathcal{U}}}, E^{\bar{\mathcal{U}}})$  with pole set  $P \subset V^{\bar{\mathcal{U}}}$  as described in step 2 in the proof of Thm. 3 with one difference: If  $\rho$  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  $\bar{U}$  as described in step 3 in the proof of Thm. 3 with one difference: If  $\rho$  is odd, the auxiliary graph  $\bar{G} = (\bar{V}, \bar{E})$  shares one 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.  $\square$

**Multi-Output support for VUCs.** An auxiliary graph that represents multi-output LUTs is a  $\Gamma_{P^+, P^-, \Omega^-}(n)$  graph as defined in Def. 12, i.e.,  $\Gamma_{P^+, P^-, \Omega^-}(n)$  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,  $\Omega^-$  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  $\Omega^-$  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  $v_{i,1}, \dots, v_{i, \Omega_i^-}$ .

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

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

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

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

*Proof.* Let  $G = (V, E) \in \Gamma_{P^+, P^-, \Omega^-}(n)$ . 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  $\deg^+(v) \leq P_i^+$  and  $\deg^-(v_{i,j}) \leq P_i^-$  from conditions 3 and 4 in Def. 12.  $\square$

We can build VUCs using Cor. 5 and UCs with constant  $\rho$  and  $\omega$  using Cor. 6, whose prove directly follows from Cors. 4 and 5.

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

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

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

*Proof.* Let  $G = (P, E) \in \Gamma_{P^+, 2, \Omega^-}(n)$  be the graph to be embedded in an EUG with  $P = \{v_{1,1}, \dots, v_{1, \Omega_1^-}, v_{2,1}, \dots, v_{2, \Omega_2^-}, \dots, v_{n,1}, \dots, v_{n, \Omega_n^-}\}$ . We can transform  $G$  into a  $\Gamma_{P^+, 2}(n + \Delta')$  graph where  $\Delta' := \sum_{i=1}^n \Omega_i^- - 1$ . Using Thm. 3, we get an EUG for  $\Gamma_{P^+, 2}(n + \Delta')$  that is bounded by

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

where  $\Delta'' := \sum_{i=1}^n \max\{\lceil \frac{P_i^+ - 2}{2} \rceil, 0\}$  and setting  $\Delta := \Delta' + \Delta''$  yields an EUG of the given size.

We need to add some more edges to the resulting EUG  $\bar{U} = (V^{\bar{U}}, E^{\bar{U}})$  with pole set  $P \subset V^{\bar{U}}$ , 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:

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

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

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

## 5.2 Applications of Varying UCs (VUCs)

If we use a VUC instead of a UC in MPC-based PFE, we get Varying Private Function Evaluation (VPFE). VPFE allows a set of  $k$  parties  $\mathcal{P}_1, \dots, \mathcal{P}_k$ , to jointly compute a circuit  $C$  held by  $\mathcal{P}_1$  on private data  $x_2, \dots, x_k$  held by  $\mathcal{P}_{i \geq 2}$  to obtain nothing but  $C(x_2, \dots, x_k)$ , and  $\mathcal{P}_{i \geq 2}$  learn nothing about  $C$  but the dimensions of all its LUTs. Thus, VPFE does not leak the whole topology of sub-circuits like SPFE (cf. §1.1), but leaks more information than PFE.

We can reduce the leakage by randomly changing the sequence of LUTs according to the topological order of the simulated circuit. In this way, building blocks (e.g., full adders) do not occur as a whole block of consecutive LUTs of the same dimension in the VUC. The function would be mapped to different sequences of dimensions 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 dimensions.

Some applications, such as logic locking (cf. [11, Fig. 3] and §1.1) do not require full privacy of the evaluated function and allow for the leakage of the sequence of dimensions of the used LUTs. However, in general PFE applications, even knowledge of the LUT sizes may reveal too much information about the protected function. Our analysis (cf. §4.3) and our benchmarks (cf. §6.3) demonstrate that many functionalities can be reduced to 3-input LUTs. Consequently, we benefit from using LUCs with 3-input LUTs in most cases. 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 dimension is used only once. Therefore, VUCs are well-suited for embedding circuits with a limited number of various LUT combinations, such as  $(3 \rightarrow 1)$ -LUT and  $(8 \rightarrow 8)$ -LUT, resulting in significant size improvements. By implementing simple functionalities with  $(3 \rightarrow 1)$ -LUTs and allowing complex functionalities with  $(8 \rightarrow 8)$ -LUTs, a wide range of possible functions can be achieved without compromising critical information (which can always be implemented using a single LUT type). The  $(8 \rightarrow 8)$ -LUTs offer a vast set of 256 combinations, enabling the implementation of a large and diverse collection of functionalities. Despite the inclusion of these additional combinations, the resulting leakage remains limited.

There are many PFE applications that benefit from such a setting, including credit checking [18], user-specific tariff calculations [22], and medical diagnosis [9]. All these applications rely on sub-functionalities such as classifiers. A classifier utilizes a mapping table 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, such as  $(8 \rightarrow 8)$ -LUTs, can efficiently implement these classifiers as they provide exactly such a table lookup. 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 computation performed to obtain the address of the lookup whose topology is hidden.

## 6 Implementation and Evaluation

We implement our proposed UC constructions using the MPC framework ABY [14] to provide a fair comparison to previous PFE works on UCs [32,23,7]. MPC frameworks supporting multi-input garbled circuits [40] reduce the communication of evaluating a single  $\rho$  input LUT to  $2^\rho - 1$  ciphertexts. In ABY, we implement  $\rho$  input LUTs as a multiplexer tree consisting of  $2^\rho - 1$  2-input AND gates, requiring  $2(2^\rho - 1)$  ciphertexts using half-gates [57]. This could be further reduced to  $1.5(2^\rho - 1)$  ciphertexts using three-halves garbling [48].

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

Building block	AND gates	XOR gates
X-switching block [36]	1	3
Y-switching block [36]	1	2
Universal Gate with $k \geq 2$ inputs	$2^k - 1$	$2^{k+1} - 2$

We benchmark our LUC construction (cf. §4) and compare it with the most recent UC of Liu et al [38] that simulates circuits with binary gates. Moreover, we evaluate our VUC construction (cf. §5) to show the improvement over Liu et al.’s UC [38] and our LUC construction. All results in this section use the EUG construction by Liu et al. [38] to construct the underlying  $T_1$  EUGs. We discuss the LUT generation in §6.1, details about our UC compilation in §6.2, and experimental results in §6.3.

### 6.1 LUT Generation

Hardware synthesis is a crucial process in electronic design automation that involves converting an abstract function description into a functionally equivalent logic implementation. This transformation is achieved through the utilization of various optimization and technology mapping algorithms. These algorithms have been extensively researched and developed over the course of many years. The resulting circuit implementation is typically dependent on the target hardware platform and the manufacturing technology employed. The two most common target hardware platforms are Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs).

This work specifically focuses on exploiting multi-input LUTs, which are fundamental components of FPGAs (which consist of logic cells containing programmable LUTs) and their corresponding synthesis tools. Although ASIC synthesis tools can also map to multi-input gates, this process is laborious, impractical, and necessitates the creation of large libraries to accommodate all possible LUTs for each input size. Thus, we chose FPGA synthesis tools. The market offers commercial FPGA synthesis tools like Intel Quartus Prime [1], VTR [2], XST [4], and Vivado Synthesis tools by Xilinx [3]. However, these tools synthesize LUT-based circuits tailored to the specific features of their respective devices. For instance, most current FPGA devices support a maximum of 6-input LUTs. In our work, we aim to generate circuits with up to 8-input LUTs, which, to the best of our knowledge, is not supported by mainstream commercial tools.

In this work, similar to [15,12], we leverage the mapping capabilities of the open-source tools Yosys [54] and ABC [10]. Yosys allows us to transform the circuit descriptions into a network of low-level logic operations represented in an intermediate format. Subsequently, ABC [10] organizes this network into a Directed Acyclic Graph (DAG) and maps it to a depth-optimized circuit composed of LUTs. It is worth noting that ABC [10] does not inherently support mapping to multi-output LUTs. To overcome this limitation, we perform post-processing on the single-output LUT circuits generated by ABC [10] and convert them into multi-output LUT circuits. Additionally, we use integrated Intellectual Property (IP) libraries within the commercial ASIC synthesis tool Synopsys Design Compiler (DC) [5], to generate circuit netlists for more complex functionalities such as floating-point operations. These circuits are initially created as Boolean netlists by Synopsys DC [5], and we subsequently remap them to LUT-based representations using the Yosys-ABC toolchain [10,54].

### 6.2 UC Compilation

Let  $C$  denote the circuit to be embedded and  $\rho$  the maximum fan-in of the circuit.

**1. Parsing the circuit:** The circuit is input in the Secure Hardware Definition Language (SHDL) [40] and parsed into the internal graph representation. We reduce the fan-out of the graph to the allowed fan-in  $\rho$  for LUCs (cf. §4) and 2 for VUCs (cf. §5) by using copy gates. For VUCs, the auxiliary graph (cf. Thm. 3) is generated. Here, we denote the auxiliary graph by  $G$  and the former graph with possibly reduced fan-out by  $\bar{G}$ .

**2. Splitting  $G$  into  $T_1$  graphs and creating  $T_1$  EUGs:** Using the LUC construction yields  $\rho$   $T_1$  graphs. For each  $T_1$  graph, we create a  $T_1$  EUG. Possible EUGs are Valiant’s EUG [52] and the 2-way split EUG of Liu et al. [38]. If we use the VUC construction, we get two  $T_1$  graphs.

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. [38] as baseline (in number of AND gates) measured with our implementations. 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  $\rho \in \{2, \dots, 8\}$  inputs and  $\omega \in \{1, \dots, 8\}$  outputs for the benchmarked circuit.

Circuit	Circuit size (# AND gates)		Improvement ( $\times$ )	LUT sizes ( $\rho \rightarrow \omega$ )
	UC of [38]	Our LUC		
AES	<b>1,779,105</b>	<b>1,779,105</b>	1.00	2 $\rightarrow$ 1
DES	1,269,537	<b>1,130,037</b>	1.12	3 $\rightarrow$ 1
MD5	3,293,262	<b>1,724,221</b>	1.91	3 $\rightarrow$ 1
SHA-1	4,872,501	<b>2,559,602</b>	1.90	3 $\rightarrow$ 1
SHA-256	10,652,234	<b>5,351,972</b>	1.99	3 $\rightarrow$ 1
Add_32	6,926	<b>3,907</b>	1.77	3 $\rightarrow$ 2
Add_64	17,006	<b>8,963</b>	1.90	3 $\rightarrow$ 2
Comp_32	2,519	<b>1,278</b>	1.97	3 $\rightarrow$ 1
Mult_32x32	347,274	<b>177,081</b>	1.96	3 $\rightarrow$ 3
Karatsuba_32x32	286,933	<b>156,888</b>	1.83	3 $\rightarrow$ 3
MD256	327,203	<b>150,046</b>	2.18	3 $\rightarrow$ 2
ED64	1,852,419	<b>947,679</b>	1.95	3 $\rightarrow$ 3
FP-Add_32	113,620	<b>90,964</b>	1.25	3 $\rightarrow$ 1
FP-Mul_32	293,125	<b>247,859</b>	1.18	3 $\rightarrow$ 1
FP-Exp2_32	2,008,269	<b>1,548,079</b>	1.30	3 $\rightarrow$ 1
FP-Div_32	372,101	<b>236,300</b>	1.57	3 $\rightarrow$ 1
FP-Sqrt_32	176,176	<b>118,873</b>	1.48	3 $\rightarrow$ 1
FP-Comp_32	6,387	<b>5,628</b>	1.13	4 $\rightarrow$ 4
FP-Log_32	1,936,813	<b>1,499,538</b>	1.29	3 $\rightarrow$ 1

**3. Edge-embedding the  $\Gamma_1$  graphs and merging them:** Each  $\Gamma_1$  graph is edge-embedded into the corresponding  $\Gamma_1$  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  $\psi$  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 [23]. Then, the  $\Gamma_1$  EUGs are merged into a  $\Gamma_\rho$  EUG (LUC) or into a  $\Gamma_2$  EUG (VUC).

**4. Basic optimizations and correctness checking:** 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 programming bits of the corresponding EUG pole. We determine the order of inputs and then set the programming bits accordingly. This also involves padding the programming bits if the gate has more inputs. Note that these additional inputs are likely to occur since each Universal Gate outputs  $\rho$  (in LUC construction) or 2 (in VUC construction) wires, independent of whether they are used in  $G$  or not. We pad the programming 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 [14]. Then, each node, along with its incoming and outgoing wires, is written into a circuit file. At the same time, the programming bits are written into a separate programming bits file.

### 6.3 Experimental Results

**Setup.** Like previous works [23,32,38], we benchmark a set of real-world circuits from [51]. In addition, we consider other useful functions like Karatsuba multiplication [29], Manhattan and Euclidean distance [15], and floating-point operations [15]. 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.

**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. As underlying MPC protocol for UC-based PFE we use Yao’s protocol [57] using free XORs [36], so XOR gates can be evaluated without communication. Therefore, we count the number of non-free AND gates to instantiate the building blocks of the UC (cf. Tab. 3). We experimentally compared our implementations with the best existing UC-based PFE construction of Liu et al. [38]. 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 [38] by 1.12 – 2.18 $\times$ .

For a comparison of the improvements in PFE, we securely evaluate our generated UCs with the GMW-based SP-LUT protocol [15] and Yao’s GC protocol [57]. 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. [38] as baseline using Yao [57] and GMW [21]. Our new UC construction is the fastest implementation: Compared to the baseline using Yao [57], 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 [57] are 1.12 – 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 by the constant round complexity of Yao’s protocol. The SP-LUT

Table 5: Runtime and communication for our LUT-based UC construction (cf. §4) compared to the state-of-the-art UC of [38] when evaluated with ABY [14]. We include the LAN evaluation time (in seconds) and the total communication (in Megabytes) between the parties in LUT-based [15], Yao sharing [57], as well as in GMW sharing [21]. The best values are marked in **bold**.

UC construction	UC of [38]				Our LUT-based UC (LUC)			
MPC protocol	Yao [57]		GMW [21]		Yao [57]		SP-LUT [15]	
Circuit	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)
AES	<b>1.811</b>	80.315	142.193	96.845	<b>1.811</b>	80.315	13.187	<b>28.427</b>
DES	1.282	57.271	101.645	68.071	<b>1.124</b>	50.570	9.233	<b>24.311</b>
MD5	3.471	148.348	238.847	168.991	<b>1.832</b>	76.638	26.642	<b>46.013</b>
SHA1	5.154	220.065	343.344	246.708	<b>2.756</b>	113.859	27.268	<b>58.641</b>
SHA-256	11.571	481.412	722.650	528.122	<b>5.878</b>	238.364	54.082	<b>123.045</b>
Add_32	0.018	0.314	1.139	0.594	<b>0.009</b>	0.177	0.224	<b>0.148</b>
Add_64	0.026	0.770	2.323	1.230	<b>0.017</b>	0.404	0.452	<b>0.319</b>
Comp_32	0.008	0.117	0.265	0.203	<b>0.004</b>	0.062	0.139	<b>0.055</b>
Mult_32x32	0.350	15.626	31.497	19.650	<b>0.212</b>	7.300	4.144	<b>4.531</b>
Karatsuba_32x32	0.292	12.901	27.214	16.597	<b>0.191</b>	6.469	3.685	<b>4.021</b>
MD256	0.337	14.801	29.037	18.326	<b>0.193</b>	6.592	4.234	<b>4.544</b>
ED64	1.924	83.552	142.147	97.558	<b>1.046</b>	39.704	17.524	<b>24.572</b>
FP-Add_32	0.164	5.105	11.780	6.859	<b>0.139</b>	4.003	2.903	<b>2.426</b>
FP-Mul_32	0.350	13.178	28.215	16.988	<b>0.308</b>	10.949	6.217	<b>4.579</b>
FP-Exp2_32	2.292	90.555	155.881	106.023	<b>1.612</b>	68.651	21.531	<b>38.330</b>
FP-Div_32	0.458	16.743	33.686	21.024	<b>0.296</b>	10.443	5.918	<b>6.528</b>
FP-Sqrt_32	0.223	7.915	18.910	10.433	<b>0.168</b>	5.237	3.417	<b>3.442</b>
FP-Comp_32	0.014	0.290	1.066	0.553	<b>0.012</b>	0.235	0.226	<b>0.118</b>
FP-Log_32	2.083	87.330	151.423	102.369	<b>1.600</b>	66.510	20.198	<b>36.287</b>

protocol [15] always has the lowest communication, achieving factor  $1.19 - 2.44\times$  less communication than Yao’s protocol. In Tab. 5, it is evident that the baseline employing Yao [57] exhibits superior runtime performance and lower total communication overhead than the baseline employing the GMW protocol [21].

Table 6: Sizes, runtime, and communication for our VUC construction (cf. §5). We include LAN evaluation times (in seconds) and total communications (in Megabytes) between the parties in LUT-based [15] as well as in Yao sharing [57]. We show the size improvement of VUC over the UC of [38] and LUC construction (cf. §4) in the last two columns. Note that VUCs reveal the LUTs’ dimensions, showcasing the enhancements obtained by sacrificing some circuit privacy.

Circuit	Size	Yao [57]		SP-LUT [15]		Size Improv. ( $\times$ )	Size Improv. ( $\times$ )
		Time	Comm.	Time	Comm.	VUC/UC of [38]	VUC/LUC
AES	1,584,047	1.724	71.753	142.221	2.607	1.13	1.13
DES	960,854	0.98	43.441	93.79	1.66	1.32	1.17
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	1.00
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	1.00
Add_64	8,963	0.02	0.40	0.45	0.32	1.90	1.00
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
MD256	112,829	0.13	5.01	1.37	4.09	2.90	1.33
ED64	947,679	1.05	39.70	17.52	24.58	1.95	1.00
FP-Add_32	90,964	0.14	4.00	2.90	2.43	1.25	1.00
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

**VUC Improvement.** Tab. 6 shows that our VUC construction which – other than LUC – leaks the fanin of the individual LUTs is up to  $2.90\times$  smaller than Liu et al.’s UC [38] when evaluated with Yao’s protocol [57], the total runtime for our sample circuits is faster by  $1.1 - 2.85\times$  and the communication is improved by  $1.06 - 2.96\times$ . This shows that significant speedups can be achieved when giving up some function privacy.

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 better size than VUC, and, if so, proceeds to compile a LUC. Nonetheless, in the majority of cases, VUC yields a better 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 dimensions, the overall size is better than VUC. However, if the circuit can be effectively constructed using LUTs with differing input dimensions, VUC performs better.

**Acknowledgements.** This project received funding from the ERC under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the DFG within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

## References

1. Intel Quartus Prime Software. <https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html>
2. Verilog to Routing. <https://verilogtorouting.org/>
3. Vivado 2023.1 - Logic Synthesis. <https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0018-vivado-synthesis-hub.html>
4. XST Synthesis. <https://www.xilinx.com/products/design-tools/xst.html>
5. Synopsys Inc. Design Compiler. <http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler> (2010)
6. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. *JoC* (1990)
7. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and Scalable Universal Circuits. *JoC* (2020)
8. Attrapadung, N.: Fully Secure and Succinct Attribute Based Encryption for Circuits from Multi-linear Maps. *Cryptology ePrint Archive*, Report 2014/772 (2016)
9. 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)
10. Berkeley Logic Synthesis and Verification Group: ABC: A system for sequential synthesis and verification, <http://www.eecs.berkeley.edu/~alanmi/abc/>
11. 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)
12. Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., Yalame, H.: FLUTE: Fast and Secure Lookup Table Evaluations. In: *S&P* (2023)
13. Cook, S.A., Hoover, H.J.: A Depth-Universal Circuit. *SIAM J. Computing* (1985)
14. Demmler, D., Schneider, T., Zohner, M.: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In: *NDSS* (2015)
15. 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)
16. Diestel, R.: *Graph Theory*. Graduate Texts in Mathematics, Springer (2010)
17. Fiore, D., Gennaro, R., Pastro, V.: Efficiently Verifiable Computation on Encrypted Data. In: *CCS* (2014)
18. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving Credit Checking. In: *ACM Conference on Electronic Commerce* (2005)
19. Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-Based Encryption for Circuits from Multilinear Maps. In: *CRYPTO* (2013)
20. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. In: *CRYPTO* (2010)
21. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: *STOC* (1987)
22. 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)
23. Günther, D., Kiss, Á., Schneider, T.: More Efficient Universal Circuit Constructions. In: *ASIACRYPT* (2017)
24. Henecka, W., Kögl, S., Sadeghi, A., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-Party Computations. In: *CCS* (2010)
25. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-Complexity Private Function Evaluation is Practical. In: *ESORICS* (2020)
26. Ji, K., Zhang, B., Lu, T., Ren, K.: Multi-party private function evaluation for RAM. *IEEE Trans. Inf. Forensics Secur.* **18** (2023)
27. 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)
28. Kamara, S., Raykova, M.: Secure Outsourced Computation in a Multi-Tenant Cloud. In: *IBM Workshop on Cryptography and Security in Clouds* (2011)
29. Karatsuba, A.A., Ofman, Y.P.: Multiplication of Many-Digital Numbers by Automatic Computers. In: *SSSR Academy of Sciences* (1962)
30. Karnaug, M.: The Map Method for Synthesis of Combinational Logic Circuits. *Transactions of the American Institute of Electrical Engineers* (1953)
31. Katz, J., Malka, L.: Constant-Round Private Function Evaluation with Linear Complexity. In: *ASIACRYPT* (2011)
32. Kiss, Á., Schneider, T.: Valiant’s Universal Circuit is Practical. In: *EUROCRYPT* (2016)
33. Kluczniak, K.: Circuit privacy for fhew/tfhe-style fully homomorphic encryption in practice. *Cryptology ePrint Archive*, Report 2022/1459 (2022)

34. Kolesnikov, V., Sadeghi, A.R., Schneider, T.: Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In: CANS (2009)
35. Kolesnikov, V., Schneider, T.: A Practical Universal Circuit Construction and Secure Evaluation of Private Functions. In: FC (2008)
36. Kolesnikov, V., Schneider, T.: Improved Garbled Circuit: Free XOR Gates and Applications. In: ICALP (2008)
37. Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant’s Universal Circuit: Improvements, Implementation, and Applications. Cryptology ePrint Archive, Report 2016/017 (2016)
38. 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)
39. Liu, Y., Wang, Q., Yiu, S.: Making private function evaluation safer, faster, and simpler. In: PKC (2022)
40. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - Secure Two-Party Computation System. In: USENIX Security (2004)
41. Masserova, E., Garg, D., Mai, K., Pileggi, L., Goyal, V., Parno, B.: Logic Locking-Connecting Theory and Practice. Cryptology ePrint Archive, Report 2022/545 (2022)
42. Patra, A., Schneider, T., Suresh, A., Yalame, H.: ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In: USENIX Security (2021)
43. Patra, A., Schneider, T., Suresh, A., Yalame, H.: SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation. In: HOST (2021)
44. Paus, A., Sadeghi, A.R., Schneider, T.: Practical Secure Evaluation of Semi-Private Functions. In: ACNS (2009)
45. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure Two-Party Computation Is Practical. In: ASIACRYPT (2009)
46. Pohle, E., Abidin, A., Preneel, B.: Poster: Fast Evaluation of S-boxes in MPC. In: NDSS (2022)
47. Quine, W.V.: The Problem of Simplifying Truth Functions. The American Mathematical Monthly (1952)
48. Rosulek, M., Roy, L.: Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits. In: CRYPTO (2021)
49. Sadeghi, A.R., Schneider, T.: Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In: ICISC (2008)
50. Shannon, C.E.: The Synthesis of Two-Terminal Switching Circuits. Bell Syst. Tech. J. (1949)
51. Smart, N., Tillich, S.: Bristol Fashion MPC circuits. <https://homes.esat.kuleuven.be/~nsmart/MPC/old-circuits.html>
52. Valiant, L.G.: Universal Circuits (Preliminary Report). In: STOC (1976)
53. Wegener, I.: The Complexity of Boolean Functions. John Wiley; Sons, Inc. (1987)
54. Wolf, C., Glaser, J., Kepler, J.: Yosys – A Free Verilog Synthesis Suite. In: Austrian Workshop on Microelectronics (2013)
55. Yao, A.C.: How to Generate and Exchange Secrets (Extended Abstract). In: FOCS (1986)
56. Yasin, M., Sengupta, A., Nabeel, M.T., Ashraf, M., Rajendran, J., Sinanoglu, O.: Provably-Secure Logic Locking: From Theory To Practice. In: CCS (2017)
57. Zahur, S., Rosulek, M., Evans, D.: Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In: EUROCRYPT (2015)
58. Zhao, S., Yu, Y., Zhang, J., Liu, H.: Valiant’s Universal Circuits Revisited: An Overall Improvement and a Lower Bound. In: ASIACRYPT (2019)
59. Zimmerman, J.: How to Obfuscate Programs Directly. In: EUROCRYPT (2015)

## A Proof of Prop. 1

*Proof.* Let  $\hat{G} = (\hat{V}, \hat{E})$  with pole set  $P \subset \hat{V}$  be the merging of the  $\Gamma_\rho(n)$  EUG  $G = (V, E)$  and the  $\Gamma_{\bar{\rho}}(n)$  EUG  $\bar{G} = (\bar{V}, \bar{E})$ . Let  $G' = (P, E') \in \Gamma_{\rho+\bar{\rho}}(n)$  be the order-preserving graph to be edge-embedded into  $\hat{G}$ . By König’s theorem [16, 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  $\psi_{\hat{G}}: E' \rightarrow \mathcal{P}_{\hat{G}}$  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 Details of Liu et al.’s EUG construction [38]

Liu et al.’s 2-way EUG construction (cf. Def. 8) is depicted in Fig. 6b and is separated into two parts:

1. We create an intermediate construction that Liu et al. [38] call *weak EUG* (lines 1-13 in Alg. 2 on page 10), which slightly differs from Valiant’s construction but does not satisfy the acyclicity condition for EUGs (cf. Def. 3). This results in the graph depicted in Fig. 6b 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 on page 10). This results in the graph depicted in Fig. 6b 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  $SP(k)_1, \dots, SP(k)_{\lceil n/k \rceil}$  (lines 3-7 in Alg. 2 on page 10). For the nested EUGs, i.e., those EUGs that are built by recursion, we use the superpole depicted in Fig. 6a.<sup>9</sup> as the recursion points of two superpoles are disjoint. 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. 6a, i.e., for  $SP(k)_i = (G_i = (V_i, E_i), P_i, \mathcal{P}_i, \mathcal{I}_i, \mathcal{O}_i)$ , the  $k$  nested EUGs in the next recursion step are built as  $G^1 = (V^1, E^1), \dots, G^k = (V^k, E^k)$  with pole sets  $P^1, \dots, P^k$ , where  $G^j \in \Gamma_1(\lceil n/k \rceil)$  and  $P^j = (\mathcal{O}_1[j], \dots, \mathcal{O}_{\lceil n/k \rceil}[j]) = (\mathcal{I}_1[j], \dots, \mathcal{I}_{\lceil n/k \rceil}[j])$  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 on page 10) for the  $k$  nested EUGs as each superpole needs a separate input/output 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  $|P_{\text{Val}^i}| = \lceil n/k \rceil - 1$  (line 12 in Alg. 1 on page 8), i.e., Liu et al.’s intermediate construction even has a worse size than Valiant’s construction. On top of that, as Fig. 6b 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. 6b 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  $v = (in/out)_i^j$  in Fig. 6b 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. 6b and lines 15-26 in Alg. 2 on page 10). 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.

<sup>9</sup> 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 on page 10 does not need the second argument (cf. Alg. 1 on page 8)

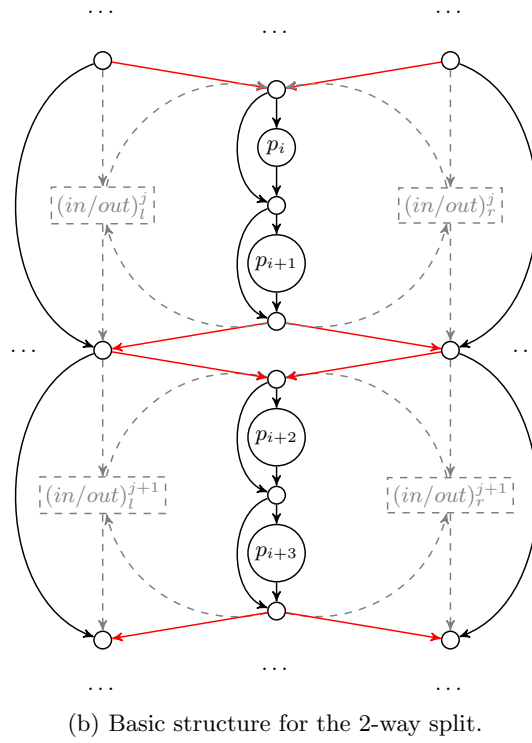
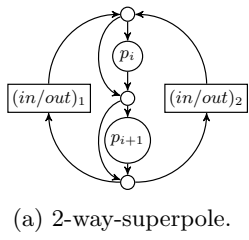


Fig. 6: (a) Superpole and (b) basic structure of Liu et al.'s 2-way split construction [38].