

A Theory of Nested Cascading in Directed Logic

Ihar Babushkin

`babushkin@iqa.uni-hannover.de`

Leibniz University Hannover

Oliver Melchert

Leibniz University Hannover

Ayhan Demircan

Leibniz University Hannover

Uwe Morgner

Leibniz University Hannover

Article

Keywords:

Posted Date: October 16th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10366269/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

A Theory of Nested Cascading in Directed Logic

Ihar Babushkin,^{1,2,3} Oliver Melchert,^{1,3} Ayhan Demircan,^{1,3} and Uwe Morgner^{1,3}

¹*Institute of Quantum Optics, Leibniz University Hannover, Welfengarten 1, 30167 Hannover, Germany*

²*Max Born Institute, Berlin, Germany*

³*Cluster of Excellence PhoenixD (Photonics, Optics,
and Engineering - Innovation Across Disciplines),*

*Leibniz University Hannover, Welfengarten 1, 30167 Hannover, Germany**

In directed logic (DL), electronically controlled optical elements serve as photonic gates. Such electro-optical elements are of mixed nature: they have two inputs—electronic and optical—but only one, optical, output. This makes cascading such gates without repeated conversion between optical and electronic representations cumbersome. This problem can be largely overcome using the nested cascading scheme proposed by Shamir and Hardy [Opt. Express, 17, 150 (2007)]. Although promising as a solution to the cascading problem, the Shamir–Hardy scheme has so far been proved or implemented only for a few simplest cases. Here, we develop a general rigorous theory of nested cascading, valid for an arbitrary number of gates. We propose a variant of the algorithm which easily extendable to large number of gates, and rigorously prove its validity. Furthermore, we analyze how nested cascading scales with the size of the corresponding Boolean formula. We show that good (linear) scalability is guaranteed in many important cases, while the average scaling with respect to the corresponding Boolean formulas is only moderately polynomial, with an exponent of approximately 3/2. Yet, the worst-case scaling remains exponential with respect to more general Boolean circuits allowing sharing and reuse of intermediate results.

INTRODUCTION

Electronic logical switches and computing devices in general are known to consume large amounts of energy per operation compared to other computational architectures—for instance, the human brain. Besides, today’s commonly used architectures suffer from the “von Neumann bottleneck” – the inherent conflict between data processing and data transfer to and from memory, resulting from the fundamentally sequential nature of such architectures. Popular approaches to overcoming these problems include delegating particular subtasks to specialized units, such as graphics processing units (GPUs), hard-coded neural networks, or specialized arithmetic logic units (ALUs).

On the other hand, during the last few years, large-scale programmable photonic integrated circuits (PICs) [2–8] have been developed. Such devices can often even be integrated directly into electronic circuits using established CMOS technology [2, 8–10]. At the same time, rapid progress in time-varying photonics [TVM]—including temporal cavity engineering [11], programmable spatio-temporal media [12], and all-optical control [13–21] has considerably expanded the range of dynamic optical functionalities available for photonic information processing.

One of these approaches is based (see Fig. 1) on optical elements such as couplers or cavities, forming a multiport optical network. Every one of these optical elements is electronically controlled using, for instance, electro-optic or thermo-optic effects. In this way, the electronic signal reconfigures an otherwise linear optical network, allowing it to perform logical operations [1, 4, 22–34] or

directly perform matrix multiplications [5, 35]. This approach has already proved its ability to overcome the overall power consumption and the dissipated power density limits compared to standard electronic units for clock frequencies beyond ~ 10 GHz [3].

The circuits performing logical operations in such a way are often gathered under the term “directed logic” (DL) [29, 31–34]. On the level of a single or a few elements, multiple physical realizations of DL have been suggested, using, as a basic element, a Mach-Zehnder interferometer Fig. 1b [26, 29] or cavities Fig. 1c coupled to propagation lines [25, 27, 32–34], electronically controlled using electro-optic [26] or thermo-optic [26, 27, 29, 32–34] effects.

When trying to connect such gates, we must remember that DL gates must be considered as devices with not only an optical input and output, but, in addition, also an electronic input (see Fig. 1a), the electronic signal determining the path taken by light, as mentioned above. The big advantage of DL is that, as soon as the electronic input is set, the processing of the optical signal takes place “with the speed of light” and, potentially, as long as losses are small, with nearly zero energy consumption [1, 22]. However, because the input contains both electronic and optical parts, whereas the output is only optical, cascading DL gates (that is, directing the output of one gate to the input of another) cannot be straightforward: for this, one needs to connect devices of different physical nature, electronic and optical. Of course, this can be achieved by converting the optical output into an electronic signal, but this operation is relatively slow and energy-intensive, and thus diminishes most of the potential power of DL. There are many possible variants of partial cascading that work in particular cases; some

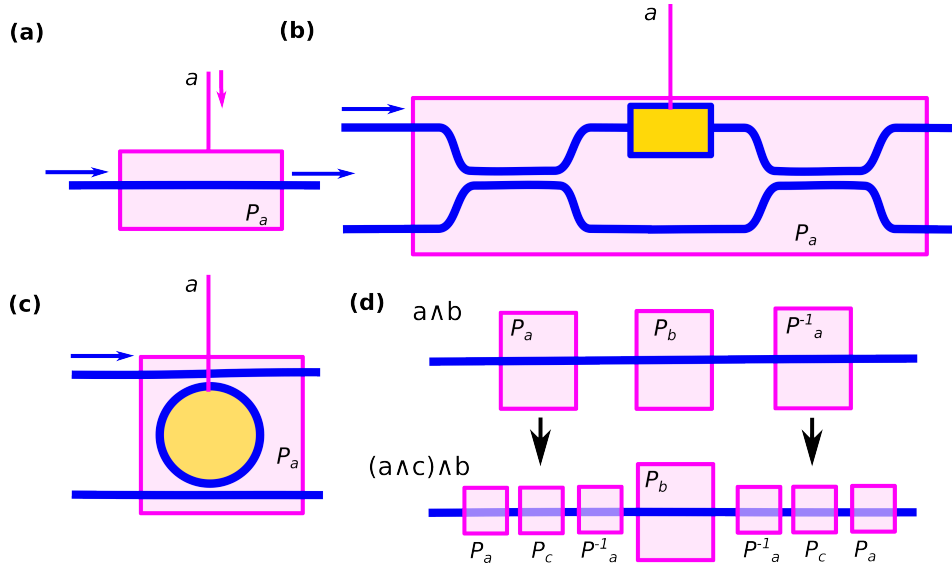


FIG. 1. (a) A basic element of DL, containing two inputs [optical (blue incoming arrow) and electronic (pink arrow)] and one optical output (blue outgoing arrow). The logical value a , supplied via the electronic channel, controls the optical path, thereby realizing an a -dependent linear operator P_a . (b,c) Exemplary realizations of a DL element P_a using a Mach-Zehnder interferometer (b) or a cavity (c). Yellow rectangles denote electro-optical or thermo-optical elements, controlled by the electronic input a . (d) General idea of nested cascading suggested by Shamir and Hardy in [1], shown by example. An exemplary logical gate $a \wedge b$ contains three basic DL elements: P_a , P_b , and P_a^{-1} (the inverse of P_a), and an exemplary cascading step $a \wedge b \rightarrow (a \wedge c) \wedge b$. In the cascading procedure, we replace P_a with the nested definition of $a \wedge c$ (see Fig. 3 for details).

of them are discussed in [36]. For instance, DL elements allow one to scalably calculate conjunctive normal forms (CNFs). Another example of partial cascading is the realization of an adder [3], where bitwise addition was performed electronically in parallel, whereas the carry bit was calculated using DL.

These examples put forward the following question: suppose we have an arbitrary Boolean function of $a, b, c, \dots$ with n variables, realized using logical gates such as OR or AND. Can we combine the corresponding optical DL gates into an optical circuit, so that, after we set all the electronic inputs $a, b, c, \dots$ (and their negations $\bar{a}, \bar{b}, \bar{c}, \dots$) of our DL elements, the circuit produces the required Boolean function at the output? This is probably the strongest form of cascability one can request from DL. In [1], Shamir and Hardy suggested a nested cascading approach to address this problem. In this approach, at every nesting step, a DL gate is replaced by a whole DL circuit implementing this gate (see Fig. 1d), giving rise to a recursive procedure. Yet, Shamir and Hardy's design for the nested-cascadable elements was rather cumbersome, denying a straightforward extension to an arbitrary number of gates; besides, it has been tested only on a few-gate configurations, and a rigorous justification and proof for arbitrarily complex cases is missing up to now.

In this article, we close this gap by developing a general theory of nested cascading. We prove rigorously that the main claim of Shamir and Hardy is indeed valid: one

can construct a nested circuit implementing an arbitrary Boolean formula. We note that, whereas AND and OR do not form a universal gate set, in the DL paradigm it is as easy to set the logical variable a as its negation $\bar{a}$. Once the negated variables are available, AND and OR gates alone are sufficient to represent any Boolean formula. Consequently, although the original proposal [1] included a larger set of gates, in this article we focus exclusively on AND and OR gates.

Furthermore, we consider the question of the scalability of nested cascading with the size of the corresponding Boolean expression. In nested logic, one of the two input variables is recursively duplicated, leading to a potentially exponential blowup. Whereas the naive approach to this question indeed suggests such a behavior [36, 37], the question is quite nontrivial because of the complex nature of nested cascading. We consider this question in detail, showing that both the worst-case and the typical scaling follow not an exponential but a power-law dependence with an exponent of $3/2$ with respect to the corresponding Boolean formula. Moreover, for a large class of Boolean expressions, the number of DL gates grows only linearly with the number of logical gates.

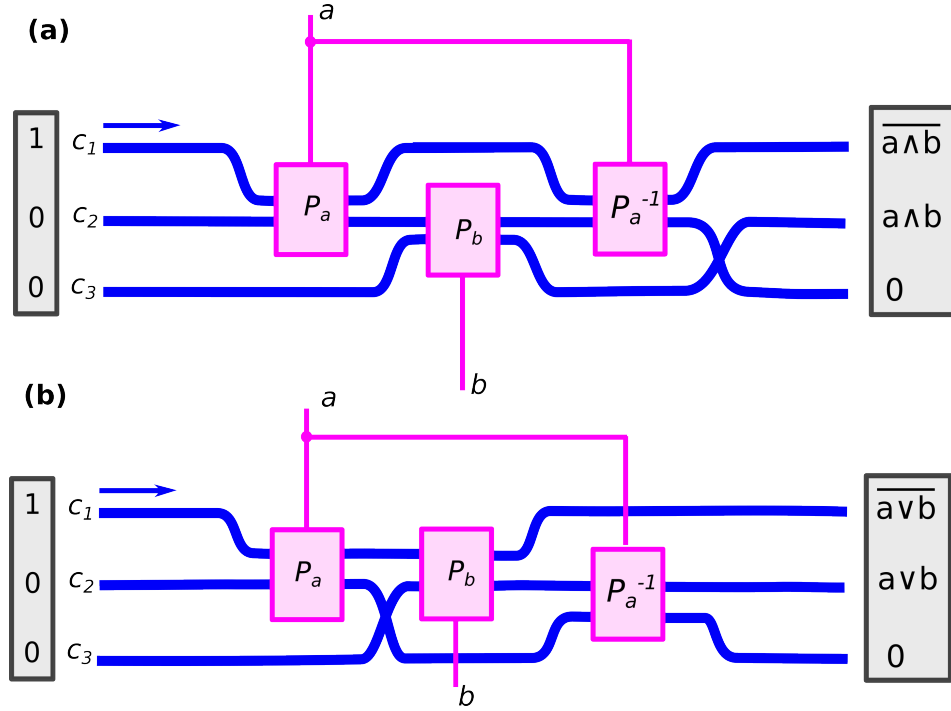


FIG. 2. Basic DL gates, representing AND (a) and OR (b). Blue lines show the optical paths, pink lines indicate the paths for the electronic signals, and pink regions show electro-optical coupling. Numbers in the gray-shaded box indicate the corresponding input and output channels when these gates are used as standalone gates (rather than as part of a larger DL circuit). Up to the channel numbering, these gates are equivalent to those in [1].

NESTED CASCADING

Formulating the procedure

In the nested cascading, suggested by Shamir and Hardy [1], the basic building blocks of the gates AND ($a \wedge b$) and OR ($a \vee b$) are the linear operators P_a , P_b and their inverses P_a^{-1} , P_b^{-1} , acting on a pair of channels (see Fig. 2(a,b), where they are represented by pink boxes), whose action is controlled by the logical variables a , b , so that for $a = 0$ the operator does nothing, whereas for $a = 1$ it swaps the two signals:

$$P_0 = \text{Identity}, \quad (1)$$

$$P_1 = \text{Swap}, \quad (2)$$

Obviously, $P_a^{-1} = P_a$. Technically, P can be realized by couplers or cavities, controlled using an electronic signal representing a or b via electro-optic [26] or thermo-optic [26, 27, 29, 32–34] effects [see Fig. 1(b,c)].

The gates $\wedge$ and $\vee$ presented in Fig. 2(a,b) are equivalent to those suggested by Shamir and Hardy in [1], even if they look somewhat different (see Supplementary for a more direct comparison). Every gate, after the electronic input is set, represents a linear optical network with 3 inputs and 3 outputs (3×3 optical multiport). Every gate assumes the optical input to be $(1, 0, 0)$ [the optical signal strength denoted by 1 can be arbitrary, but must, of

course, be the same for all gates]. The output vector has the same energy as the input, which is a consequence of unitarity.

Depending on the values of a and b , the optical signal from channel 1 is redirected either to the first output channel [this happens if $\overline{a} \wedge \overline{b} = 1$ for the AND gate in Fig. 2(a) and if $\overline{a} \vee \overline{b} = 1$ for the OR gate in Fig. 2(b)] or to one of the two remaining channels in the opposite case. That means that, if we look at the first output, it “calculates” the logical expression $\overline{a} \wedge \overline{b}$ or $\overline{a} \vee \overline{b}$. Other outputs calculate different logical expressions—the full truth tables for all outputs are given in Appendix A (Fig. 9).

In [1], furthermore, it was described how these basic gates can be combined with each other, representing arbitrary Boolean formulas. To do this, Hardy and Shamir suggested a nested approach. To create the gate $(a \vee c) \wedge b$, one should take the DL circuit for $a \wedge b$ and replace P_a with the DL circuit for $a \vee c$ and P_a^{-1} with the DL circuit for $a \vee c$, only reversed (see Fig. 3a,b). To obtain $c \wedge (a \vee b)$, they suggested taking the DL circuit for $c \wedge b$ and replacing P_b with $a \vee b$ (see Fig. 3c,d).

Despite a very promising proposal, the authors checked their approach only on a few examples. They also did not provide any proof or truth tables. Besides, even if we assume it works, from the way they presented the circuits it is difficult to understand how exactly they can be scaled to a larger number of gates. Indeed, the pro-

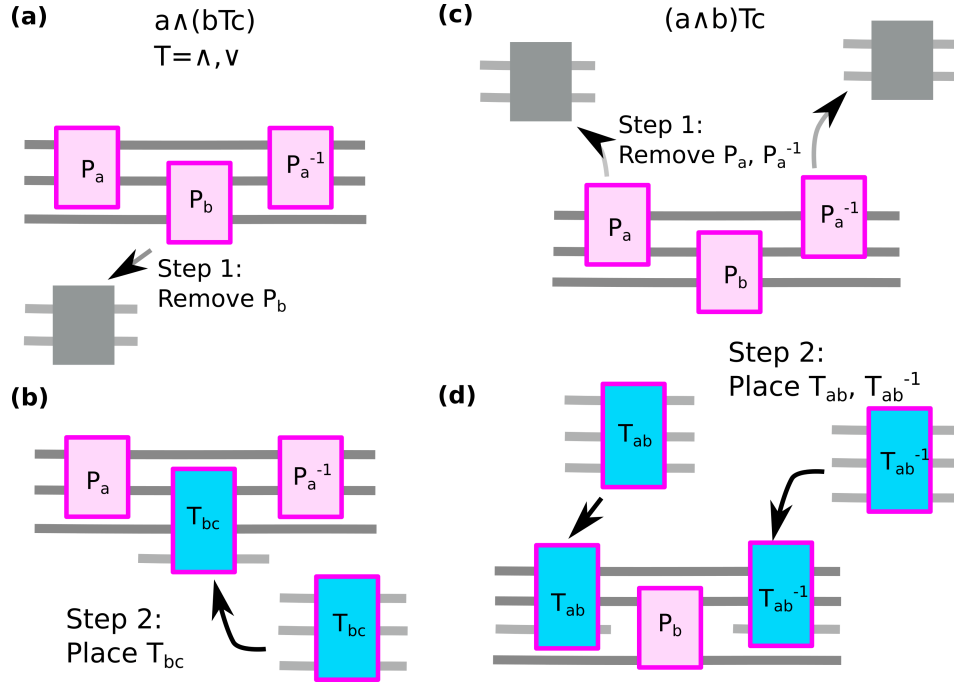


FIG. 3. Nested cascading of DL gates [1]. The cascading procedure applied at the lowest level of the circuit to obtain $a \wedge b \rightarrow a \wedge (b T c)$ (a,b), where $T \in \{\wedge, \vee\}$, or $a \wedge b \rightarrow (a T b) \wedge c$ (c,d). The last input and output channels are always left unused, while the remaining two inputs and outputs are connected in an order-preserving manner. An analogous procedure is valid for $a \vee b \rightarrow a \vee (b T c)$ and $a \vee b \rightarrow (a T b) \vee c$. For simplicity, we denote P_{xTy} by T_{xy} for $x, y \in \{a, b, c\}$. Note that the last slot in every gate must be replaced by the inverse operator $P_{xTy}^{-1} = T_{xy}^{-1}$.

cedure suggests that we replace a 2-port device P_a, P_b with 3-port devices (representing AND or OR). This replacement leaves one of the inputs and one of the outputs “hanging” (see Fig. 3), and from the discussion in [1] it is not clear at all which of the inputs and which of the outputs it should be. Moreover, the overall order of the inputs and outputs by this substitution is not fixed in [1]. Here, we adopt a gate design for which the corresponding rules are particularly simple: the last input and output channels are always left unused, while the remaining two inputs and outputs are connected in an order-preserving manner, with the lower-numbered input connected to the lower-numbered output. We note that the resulting circuits are not always topologically equivalent to those presented in Ref. [1] (see Supplementary Information).

Keeping this in mind, we can construct a rigorous approach to convert any Boolean formula L , which contains AND and OR gates, into a DL circuit P_L :

- Convert the Boolean formula L into a binary tree, that is, into a form in which every gate has fan-in 2 (see Fig. 5). In doing so, avoid interpreting configurations of the form $\dots a \wedge b \wedge c \dots$ as $(a \wedge b) \wedge c$, and instead use $a \wedge (b \wedge c)$; the particular configuration $(a \wedge b) \wedge c$ is not allowed (the reason for this rule will be explained later).
- Select the gate T at the root of the tree and con-

struct the corresponding DL circuit for $a T b$ as shown in Fig. 2.

- Proceed recursively down the tree, successively following the left and right branches. Every such step can be described as $a T b \rightarrow (a T_1 b) T c$ if we select the left branch, or as $a T b \rightarrow a T (b T_1 c)$ if we select the right branch, for some T, T_1 . Transfer this step to the DL circuit by applying the procedure shown in Fig. 3. In particular, when replacing the 2-port elements P_x with a 3-port gate, connect the first two outputs in an order-preserving manner and keep the last one “hanging,” meaning that at this point a new channel is created.

This procedure ensures that at every step we replace the 2-port devices P_x with 3-port devices exactly as shown in Fig. 3.

After the DL circuit has been constructed, we run it as follows:

- Set the operators P_x according to the Boolean variables x as given in Eqs. (1)-(2).
- Apply the input vector $(1, 0, 0, \dots, 0)$ to the circuit (meaning a nonzero signal in the first channel and zeros in all remaining channels), see Fig. 4.
- Read the output from either the first channel, c_1 , or the second channel, c_2 . The first channel con-

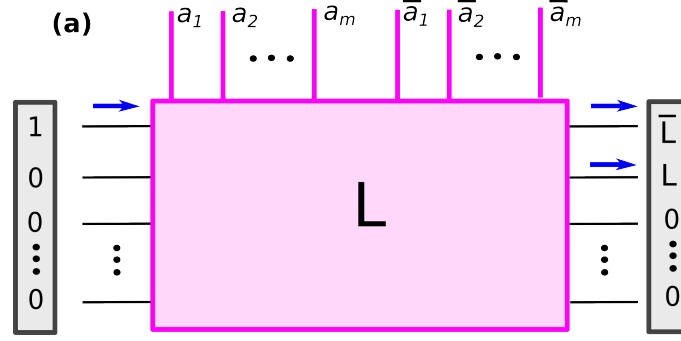


FIG. 4. A general DL circuit representing $L(a_1, \bar{a}_1, a_2, \bar{a}_2, \dots)$ after cascading. After the variables a_i and $\bar{a}_i$ are set, the entire circuit becomes a linear operator that transforms an input vector into an output vector. The gray boxes show the valid input and the corresponding output vectors.

tains $\bar{L}$, that is, zero if $L = 1$ and nonzero if $L = 0$, whereas the second channel contains L . All remaining output channels contain zeros.

In Fig. 6, a DL circuit is shown for the simple formula $L = (a \wedge b) \vee c$, together with its truth table, representing the values of all outputs for all values of the logical variables a, b, c . All other DL circuits with 1 and 2 gates are presented in Appendix A (Figs. 10-16). A more complex exemplary circuit for $L = (((a \wedge b) \vee c) \wedge d) \vee (e \wedge f)$ is given in Fig. 7.

Close comparison (see Supplementary) shows that the connectivity topology resulting from this procedure is not fully equivalent to that of Hardy and Shamir. The differences, however, are presumably confined to paths that are never traversed.

As we will see in the next subsection, this property of “having a $\bar{L}$ -output in c_1 ” and “having a L -output in c_2 ” is transferred from 1- and 2-gate expressions to expressions with an arbitrary number of gates.

Universality proof by induction

The procedure described above suggests that for any Boolean circuit L we can construct an equivalent DL circuit P_L , which calculates $\bar{L}$ and L at the output of the first two channels c_1, c_2 . For small circuits this can be tested directly, which we did for all 1- and 2-gate circuits (see Table I and Figs. 9-16 in Appendix A, as well as Fig. 6). We cannot make such a direct test for an arbitrary number of gates; therefore, a proof is necessary. We prove this statement by mathematical induction: assuming that for step n (in the procedure described above) the property we want to prove (the induction hypothesis) holds, we prove it for step $n + 1$. For $n = 1$, the proof is provided by direct verification.

Before we start, let us formulate more carefully the property we want to prove. We observe that there are in fact two separate independent properties to be proved:

(i) We can map every logical circuit L into a DL circuit which evaluates L , and (ii) We can map the logical refinement step (such as $a \text{ T}_1 b \rightarrow (a \text{ T}_2 b) \text{ T}_1 c$ for some gates $\text{T}_i, i = \{1, 2\}$) into the corresponding refinement step in the DL circuit. With this in mind we formulate i) and ii) more formally, as two induction hypotheses:

(H1) Circuit mapping. For every Boolean expression

$$L(a_1, a_2, \dots, a_m),$$

constructed from the variables a_j using the operations $\wedge, \vee$, the nesting algorithm described above produces a DL-circuit

$$P_{L(a_1, \dots, a_m)}$$

such that, for any assignment of the electronic controls $a_j \in \{0, 1\}$, the first optical output of the circuit evaluates the Boolean function $\bar{L}$ whereas the second optical output of the circuit evaluates the Boolean function L , and all other outputs contain zeros.

(H2) Refinement mapping. Let $L(a_1, \dots, a_k, \dots, a_m)$ be an arbitrary Boolean expression, and let one of its variables a_k be replaced by a depth-1 Boolean subexpression,

$$a_k \mapsto c_1 T c_2, \quad T \in \{\wedge, \vee\}.$$

Then the corresponding DL circuit undergoes the transformation

$$P_{L(a_1, \dots, a_k, \dots, a_m)} \longrightarrow P_{L(a_1, \dots, (c_1 T c_2), \dots, a_m)},$$

by replacing the physical 2-port device P_{a_k} (and the corresponding inverse $P_{a_k}^{-1}$) with the 3-port DL realization of the gate $c_1 T c_2$, according to the nesting rules described above. The resulting circuit evaluates the refined Boolean expression.

After formulating the induction hypotheses (H1) and (H2), we can continue with their proof. First, we verify (H1) and (H2) directly for $n = 1$, i.e., for the simplest

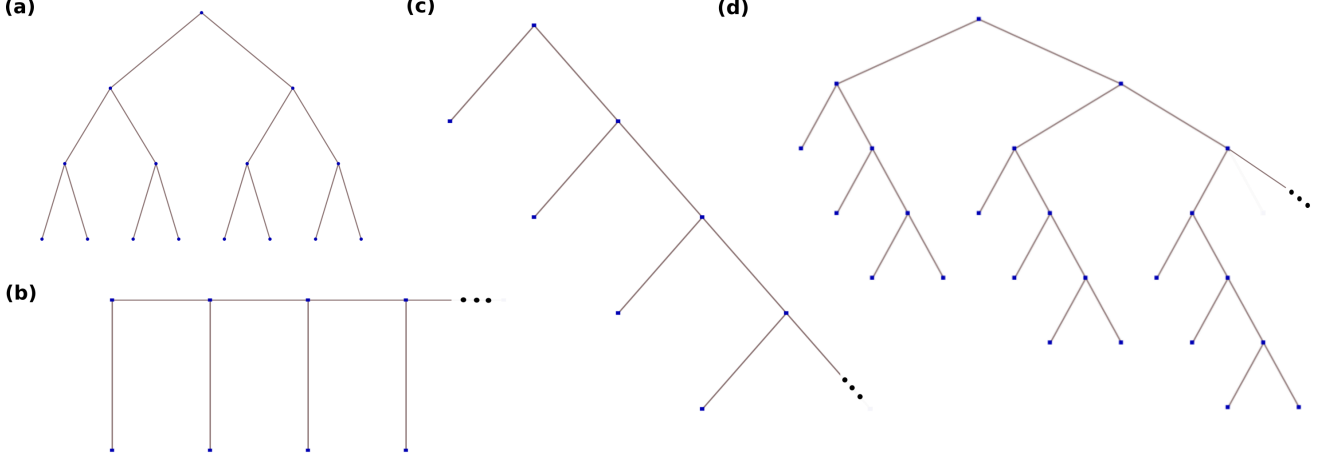


FIG. 5. Some examples of Boolean formulas represented as trees. Terminal nodes represent logical variables, whereas all other nodes represent gates. (a) Full-depth circuit consisting of fan-in 2 gates. (b) An example of a shallow circuit of depth 1, for instance $a \wedge b \wedge c \dots$, with gates having large fan-in (the horizontal line with many nodes represents a single gate in this case). (c) Fan-in 2 representation of the shallow circuit in (b), for instance $a \wedge (b \wedge (c \dots))$. (d) A circuit of depth 2 (such as CNFs and DNFs) in fan-in 2 representation.

Boolean formula	c_1	c_2	c_3	c_4	c_5
$a \wedge b$	$\overline{a \wedge b}$	$a \wedge b$	0		
$a \vee b$	$\overline{a \vee b}$	$a \vee b$	0		
$a \vee (b \wedge c)$	$\overline{a \vee (b \wedge c)}$	$a \vee (b \wedge c)$	0	0	
$(a \vee b) \wedge c$	$\overline{(a \vee b) \wedge c}$	$(a \vee b) \wedge c$	0	0	0
$a \wedge (b \vee c)$	$\overline{a \wedge (b \vee c)}$	$a \wedge (b \vee c)$	0	0	
$(a \wedge b) \vee c$	$\overline{(a \wedge b) \vee c}$	$(a \wedge b) \vee c$	0	0	0
$a \wedge (b \wedge c)$	$\overline{a \wedge (b \wedge c)}$	$a \wedge (b \wedge c)$	0	0	
$(a \wedge b) \wedge c$	$\overline{(a \wedge b) \wedge c}$	$(a \wedge b) \wedge c$	0	0	0
$a \vee (b \vee c)$	$\overline{a \vee (b \vee c)}$	$a \vee (b \vee c)$	0	0	
$(a \vee b) \vee c$	$\overline{(a \vee b) \vee c}$	$(a \vee b) \vee c$	0	0	0

TABLE I. Truth tables for the outputs c_i of the basic direct circuits. The truth tables are given in the condensed form of Boolean functions producing the same outputs for all values of a , b , and c . The complete truth tables, as well as graphical representations of the corresponding circuits, can be found in Appendix A (Figs. 9-16) and Fig. 6. Note nonassociativity of DL. It can be seen that c_1 always contains the negation of the corresponding Boolean formula, whereas c_2 contains the formula itself. All other outputs contain zeros.

possible case, in order to establish the basis for the induction. Specifically, we construct a single gate [step (H1)] and extend it to all possible two-gate configurations [step (H2)]. In other words, proving the case $n = 1$ requires checking all possible single- and two-gate configurations. This is done in Table I, which lists, for every channel, the corresponding Boolean expression, and in Appendix A (Figs. 9-16), where the corresponding circuits and the values of all output channels are given explicitly. Table I verifies that in all the simplest cases, (H1) and (H2) hold.

Having verified the case $n = 1$, we proceed by induction. We assume that (H1) and (H2) hold for all Boolean expressions whose nesting depth does not exceed n . Let $L(a_1, \dots, a_m)$ be such an expression. Consider a refinement of the expression obtained by replacing an arbitrary

variable a_k inside L by a binary subexpression $c_1 T c_2$. By (H1), the DL circuit $P_{L(a_1, \dots, a_m)}$ correctly evaluates L . Applying a single nesting operation—namely, replacing the controlled 2-port device P_{a_k} and its inverse by the 3-port DL realization of $c_1 T c_2$ —produces a new circuit. By construction, this circuit is exactly $P_{L(a_1, \dots, (c_1 T c_2), \dots, a_m)}$ and, by (H2), correctly evaluates the refined Boolean expression. Thus, both hypotheses remain valid for expressions of nesting depth $n + 1$, thereby completing the proof.

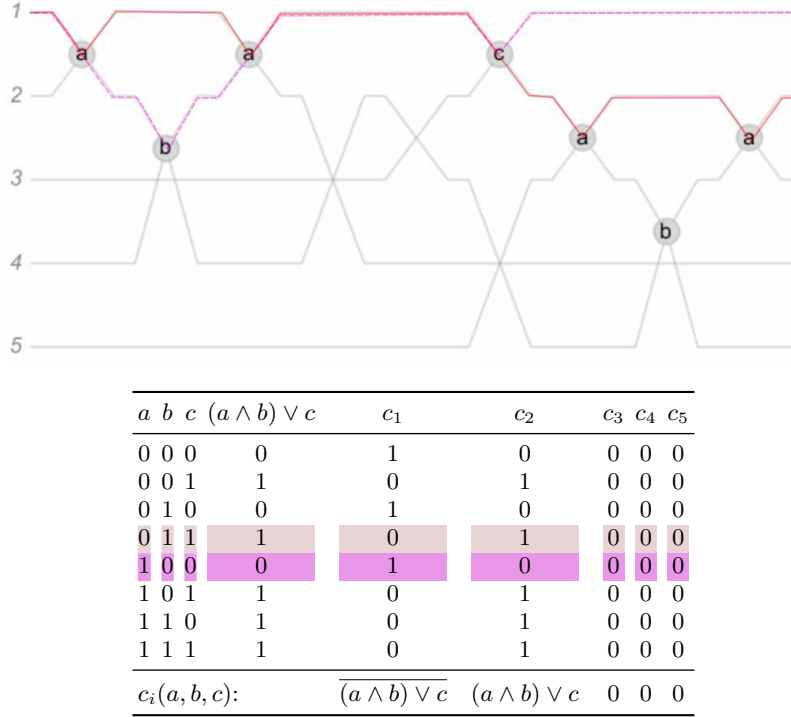


FIG. 6. Exemplary DL circuit representing $L = (a \wedge b) \vee c$ (upper panel) and the corresponding truth table (lower panel) for each of the outputs $c_i(a, b, c)$. DL is made as a composition of two gates $x = a \wedge b$ and $x \vee c$ prepared according to the rule Fig. 3c,d; the rectangles show sub-gates $a \wedge b$. For the graphical representation, we use compact notation for the optical elements. Namely, every circle labeled with $x = a, b, \dots$ represents the element P_x according to Eqs. (1)-(2). Note that $P_x^{-1} = P_x$, so both direct and iverse elements are represented by the same symbols. The colored lines represent the light paths for two exemplary cases: $a = 0, b = 1, c = 1$ (solid pale pink line), corresponding to $L = 1$, and $a = 1, b = 0, c = 0$ (magenta dashed line), corresponding to $L = 0$. For convenience, the table also contains, together with the values of a, b, c , the values of the function $L(a, b, c)$. The lowest row of the table summarizes the outputs $c_i(a, b, c)$ in the form of Boolean functions. The cases corresponding to the exemplary light paths in the circuit above are highlighted using the corresponding colors.

SCALABILITY OF THE PROCEDURE

As shown in the previous sections, nested cascading enables the implementation of any Boolean formula (represented as a tree) using DL circuits. At this point, it is important to distinguish between Boolean formulas and the more general concept of Boolean circuits. Some examples of Boolean trees are shown in Fig. 5. In a Boolean tree, every node has exactly one parent, and every intermediate result is therefore used only once. This is precisely the property that distinguishes Boolean formulas from general Boolean circuits. By contrast, a Boolean circuit is a directed acyclic graph, in which intermediate results may be shared and reused by multiple subsequent gates. General Boolean circuits cannot be realized by cascading DL elements alone, since reusing an intermediate result requires converting the corresponding optical signal back into an electronic representation before it can control subsequent DL gates. The natural question is then how efficiently Boolean formulas can be realized using nested DL circuits. More precisely, we investigate how the number of optical elements, N_{opt} , scales with

the number of logical gates, N_{bool} , for several practically important classes of Boolean formulas.

The basic gates in Fig. 2 contain two operators P for the first argument a . This leads to a risk that at every step of expansion the size of the device will double (see Fig. 1d), leading to an exponential growth of N_{opt} with N_{bool} . This happens in particular if at every expansion step we add new gates into this “bad” slot a . Can we avoid this by a clever arrangement of the Boolean gates?

This is indeed always the case for shallow circuits of depth 1 such as those presented in Fig. 5b. When reformulating such circuits into fan-in 2 form, as required by our procedure (Fig. 5c), we can always put the continuation of the circuit into the “good,” non-duplicating slot b , whereas the “bad,” duplicating slot appears only in the terminal nodes (variables). That is, the Boolean expression $a \wedge b \wedge c \dots$ must be converted into the fan-in 2 form as $a \wedge (b \wedge (c \dots))$ rather than as $((\dots (a \wedge b) \wedge c) \dots)$ (and similarly for $\vee$; for $\wedge$ chains, in addition, the configuration $(a \wedge b) \wedge c$ is additionally prohibited, as discussed above). For the depth-1 circuit shown in Fig. 5b, we therefore obtain only a constant overhead,

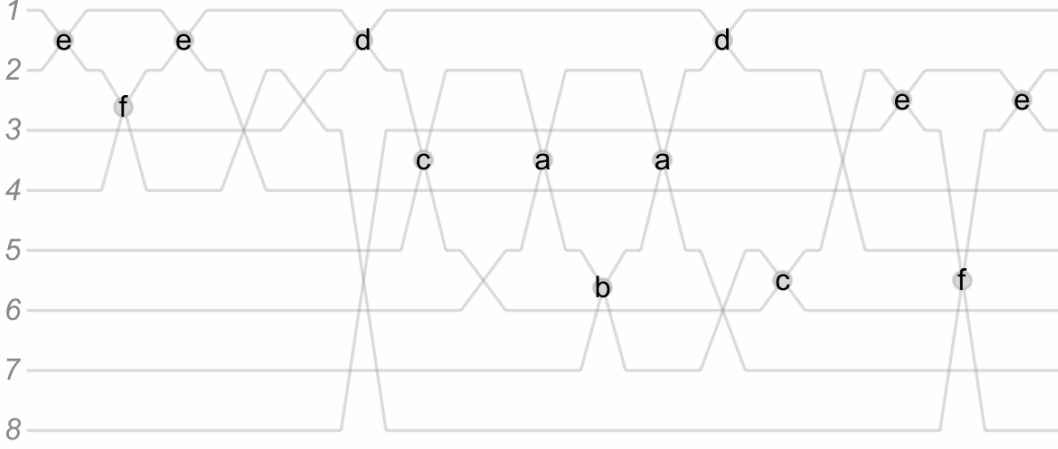


FIG. 7. An exemplary DL circuit for $L = (((a \wedge b) \vee c) \wedge d) \vee (e \wedge f)$ realized in the optimal form with the right-branching (Fig. 3a,b) was used whenever possible [$L = ((e \wedge f) \vee (d \wedge (c \vee (a \wedge b))))$]. According to the general procedure described above, the input is $c_1 = 1$, $c_i = 0$ for $i > 1$. The main output channels are c_1 and c_2 , containing $\bar{L}$ and L , respectively. All other output channels contain zeros.

$$N_{\text{opt}} \sim N_{\text{bool}}. \quad (3)$$

Will we observe something similar in more complex cases? Let us first focus on a few representative classes of Boolean expressions: conjunctive and disjunctive normal forms (CNFs and DNFs) (see Fig. 5d), fully nested binary trees (Fig. 5a) with alternating logical operations, and binary adders. Importantly, in the scalings $N_{\text{opt}}(N_{\text{bool}})$ below we compare the DL circuits to the corresponding Boolean formulas (which were used to obtain these DL circuits) rather than to more general and potentially more optimal Boolean circuits.

Conjunctive and disjunctive normal forms

A CNF is a Boolean expression of the form

$$L_{\text{CNF}} = \bigwedge_{i=1}^m \left(\bigvee_{j=1}^{k_i} x_{ij} \right), \quad (4)$$

where x_{ij} are Boolean variables (possibly negated). After converting all logical operations to fan-in-2 form (cf. Fig. 5d), the number of Boolean gates scales as $N_{\text{bool}} \sim \sum_i k_i$.

For CNFs, the nested DL construction exhibits particularly favorable scaling, as follows from Fig. 5d. Indeed, each OR clause by itself is a depth-1 circuit and demonstrates only linear scaling according to Eq. (3). As we proceed down the tree in Fig. 5d, we must add every OR subtree into the doubling slot only once. That is, effectively, our procedure is equivalent to doubling the

OR subtrees, which leads to only a constant overhead, preserving the linear scaling for this case:

$$N_{\text{opt}} \sim N_{\text{bool}}. \quad (5)$$

And indeed, this reasoning is fully supported by numerical simulations. In our numerical simulations, we calculated $N_{\text{opt}}(N_{\text{bool}})$ using random CNF forms of increasing size, controlled by the increasing parameter n with $k_i = n$ and $m = n$ in Eq. (4), for n up to $n_{\text{max}} = 20$. We indeed found the linear scaling

$$N_{\text{opt}} \approx -20.80 + 5.69 N_{\text{bool}} \quad (6)$$

as given by the fit (see Fig. 8). The reasoning above confirms that shallow Boolean circuits with a fixed small depth can be implemented efficiently within DL. Exactly the same reasoning is also applicable to DNFs (which have the same structure as CNFs).

Fully nested binary trees

As a further important example, we consider fully nested Boolean expressions (see Fig. 5a) with alternating logical operations, represented as complete binary trees, for instance

$$L_{\text{tree}} = ((x_{11} \vee x_{12}) \wedge (x_{21} \vee x_{22})) \vee ((x_{31} \vee x_{32}) \wedge (x_{41} \vee x_{42})), \quad (7)$$

and recursively extended to larger depths.

In such expressions, both inputs of a logical gate are themselves nontrivial subexpressions. This is the most unfortunate situation for the nested DL construction, since it implies that at each level, one of the subcircuits comes into the doubling slot of a DL gate and must inevitably be duplicated. Thinking naively, this should

lead to an exponential scaling of N_{opt} with N_{bool} , but this does not happen. Numerically, we find a power-law dependence

$$N_{\text{opt}} \approx a N_{\text{bool}}^k, \quad a \approx 1.55, \quad k \approx 1.56, \quad (8)$$

see Fig. 8, where we considered tree depths up to $n = 7$.

This power-law scaling can be understood analytically. For this, let us consider a depth-2 tree. Without duplication, the number of elements on the lower level would be twice as large (by the very nature of a binary tree). With the duplication of one of the slots, the number of elements becomes three times larger instead. Therefore, we conclude that

$$N_{\text{opt}} \sim 3^n, \quad (9)$$

whereas for the Boolean gates we have obviously

$$N_{\text{bool}} \sim 2^n. \quad (10)$$

Expressing N_{opt} as a function of N_{bool} from Eqs. (9)-(10), we obtain

$$N_{\text{opt}} \sim N_{\text{bool}}^{3/2}. \quad (11)$$

This analytical scaling is quite close to the one obtained numerically (see Fig. 8 and Eq. (8)), where the dashed line represents the power law $N_{\text{opt}} = 2N_{\text{bool}}^{3/2}$.

Adder circuit

In this subsection we consider the Boolean realization of an n -bit ripple-carry adder, which computes the sum of two binary numbers $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$, where a_1 and b_1 denote the least significant bits. The adder produces a sequence of sum bits $(s_1, \dots, s_n)$ and a final carry-out bit c_{n+1} . The computation proceeds sequentially from the least significant to the most significant bit, with each bit depending on the corresponding input bits and the incoming carry.

For each bit position i , the sum bit is given by

$$s_i = a_i \oplus b_i \oplus c_i, \quad (12)$$

where c_i is the carry into the i -th bit. The initial carry is set to $c_1 = 0$. The carry bits are then generated recursively according to

$$c_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i), \quad c_1 = 0. \quad (13)$$

Here we focus on the carry-out bit c_{n+1} .

In this case, the numerically observed scaling (for $n \leq 7$) again follows a power law,

$$N_{\text{opt}} \approx a N_{\text{bool}}^k, \quad a \approx 0.73, \quad k \approx 1.55. \quad (14)$$

The exponent is close to that obtained for the fully nested tree, whereas the prefactor a is smaller, so that N_{opt} for

the adder is strictly smaller than N_{opt} for the full binary tree. This is not accidental, since the tree representing the adder is obviously a subtree of the full binary tree. This relation is discussed in more detail in the next subsection.

It is important to emphasize that if the adder were implemented as a Boolean circuit, allowing intermediate results to be shared and reused in recursive expressions such as Eq. (13), it would exhibit linear scaling, $N_{\text{bool}} \sim n$. We, however, consider N_{bool} for the corresponding Boolean formula (rather than a circuit); when treating c_i as a Boolean formula, the carry recursion in Eq. (13) involves repeated substitution of c_i into c_{i+1} , which leads to exponential growth of N_{bool} with n . For DL circuits, sharing and reuse are also not available. That is, for the adder in Eq. (14), both N_{bool} and N_{opt} are exponential in n , yet N_{opt} grows only polynomially with N_{bool} as given by Eq. (14).

Scalings for worst and most typical cases

Returning to the question of what the “worst,” “general,” or “most typical” scaling $N_{\text{opt}}(N_{\text{bool}})$ is, we notice that, in fact, the full binary tree represents the most typical type of Boolean formula. Indeed, most Boolean formulas are full binary trees (cf. Fig. 5a) with some arrangement of gates at the nodes. The remaining Boolean formulas are binary trees with some of the edges removed. Therefore, the power-law scaling Eq. (11) must be considered very close to the worst-case scaling and, at the same time, as the most typical scaling. The case of the adder considered above is a very good illustration of this point: the adder is obviously “smaller” than the full tree, and therefore its scaling must be milder, which we also readily observe in Fig. 8.

Note, however, that this scaling, as already mentioned, represents the relation between the DL circuit size and the size of the corresponding Boolean formula, which, in turn, may exhibit scaling that is exponentially worse than that of a Boolean circuit, which allows sharing and reuse of intermediate results, as in the case of the adder discussed above. On the other hand, as we see in Fig. 7, quite a large part of the circuit can be removed since it does not influence the value in the “main” channel c_1 . Consideration of the influence of such optimizations on the scaling is, however, beyond the scope of this paper.

CONCLUSIONS AND DISCUSSION

In this work, we have revisited the nested cascading in directed logic, proposed by Hardy and Shamir in [1]. Despite its great promise, the ingenious idea of Hardy and Shamir has been up to now neither properly proved

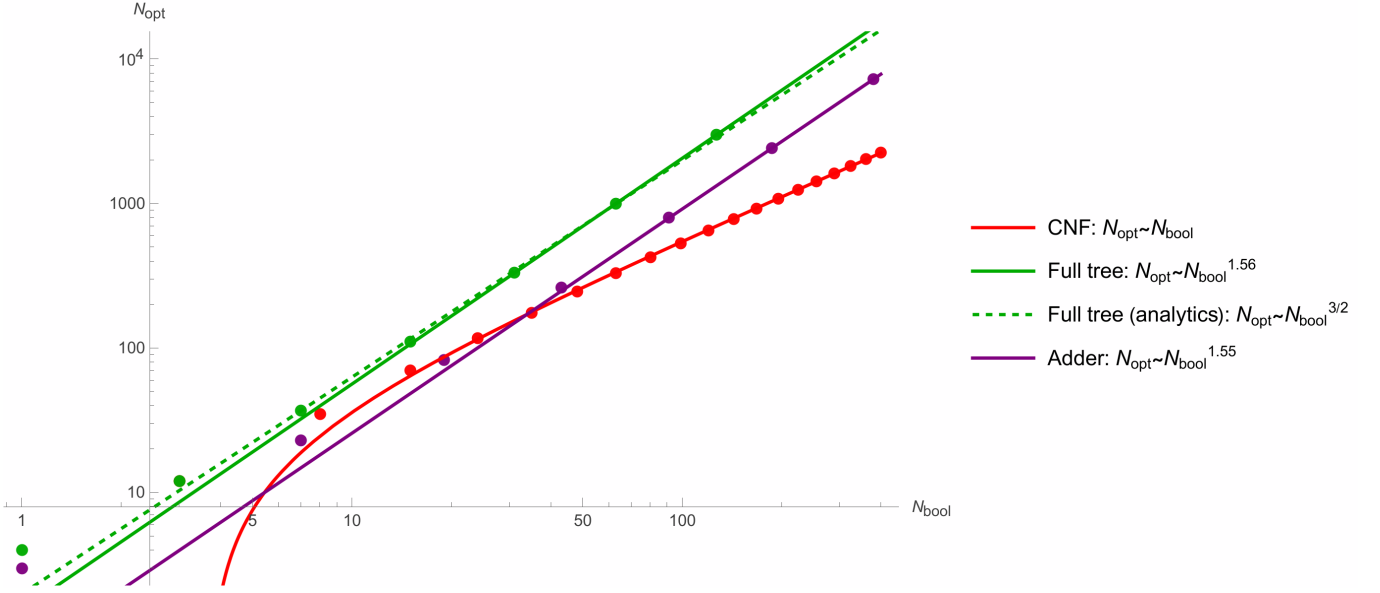


FIG. 8. Scaling of the optical gate count N_{opt} as a function of the Boolean gate count N_{bool} for different classes of Boolean formulas realized using nested directed logic. Symbols show numerically calculated data points, whereas the solid lines represent fits to these data. The dashed line shows the analytically obtained “most typical” power-law scaling $N_{\text{opt}} \sim N_{\text{bool}}^{3/2}$. Red symbols show conjunctive normal forms (CNFs), green symbols correspond to a full binary tree, whereas purple symbols represent the carry-out bit of an n -bit ripple-carry adder.

nor thoroughly tested beyond some of the few-gates configurations.

We filled these gaps by constructing an unambiguous cascading procedure and providing a rigorous proof of its correctness. We showed that the main promise of this approach does indeed hold: for every Boolean formula L , consisting of AND and OR operations and converted into fan-in-2 form, one can construct, under certain composition rules, a DL circuit that evaluates both L and $\bar{L}$. In this article, we focus only on AND and OR gates, since, once the negated variables are available, these gates alone are sufficient to represent any Boolean formula.

Our proof is based on mathematical induction. By its very nature, an inductive proof derives the validity of arbitrarily complex gates from the validity of elementary gates and their simplest compositions, provided that the latter are verified directly. This direct verification included 72 different configurations, all of which were tested by us.

In general, for purely optical DL circuits, without back-conversion of optical signals into electronic ones, sharing and reuse of intermediate results are not possible. Boolean formulas, in contrast to the more general Boolean circuits, also do not allow such sharing. As a consequence, the size of a Boolean formula can, in some cases, be exponentially larger than that of the corresponding optimal Boolean circuit.

On the other hand, even when comparing the nested DL implementation with the corresponding Boolean formula, one might expect a further exponential increase

in the number of DL elements, since the nested DL implementation requires one of the arguments to be duplicated. We analyzed this question and showed that both the worst-case and the typical scaling of the number of DL elements relative to the corresponding Boolean formula are not exponential, but are asymptotically given by a power law with an exponent close to $(3/2)$. Similar scaling is observed for important algorithms such as addition.

Thus, the nested DL construction faithfully reflects the complexity of Boolean formulas, or trees, rather than that of Boolean circuits. The size of DL circuits is generally at most polynomially larger than that of the corresponding Boolean formula, while both can be exponentially larger than the corresponding optimal Boolean circuit. However, there are important cases, such as conjunctive and disjunctive normal forms, in which the DL circuit scales linearly with both the corresponding Boolean formula and the corresponding Boolean circuit.

In general, the difficulty of DL logic in comparison with optimal Boolean circuits is not the duplication of variables, but rather the lack of sharing and reuse of intermediate results. Our results also indicate that the points at which such sharing and reuse become necessary are the most natural places to go beyond pure DL cascading and use mixed architectures, including back-conversion from optical to electronic signals, optical nonlinearities, or other nonlinear mechanisms.

Appendix A: truth tables and graphical representations of 1- and 2-operator Boolean formulas

In this appendix, the truth tables and the graphical representation of all two-variable expressions $a \mathsf{T} b$, $\mathsf{T} \in \{\wedge, \vee\}$ as well as three-variable expressions $(a \mathsf{T}_1 b) \mathsf{T}_2 c$ and $a \mathsf{T}_1 (b \mathsf{T}_2 c)$, $\mathsf{T}_i \in \{\wedge, \vee\}$, $i = 1, 2$ in Figs. 9-16. Note that the circuit and the truth table for the case $(a \wedge b) \vee c$ are already shown in Fig. 6 and therefore are not given in this Appendix. Denotations for all the figures here are as in Fig. 6.

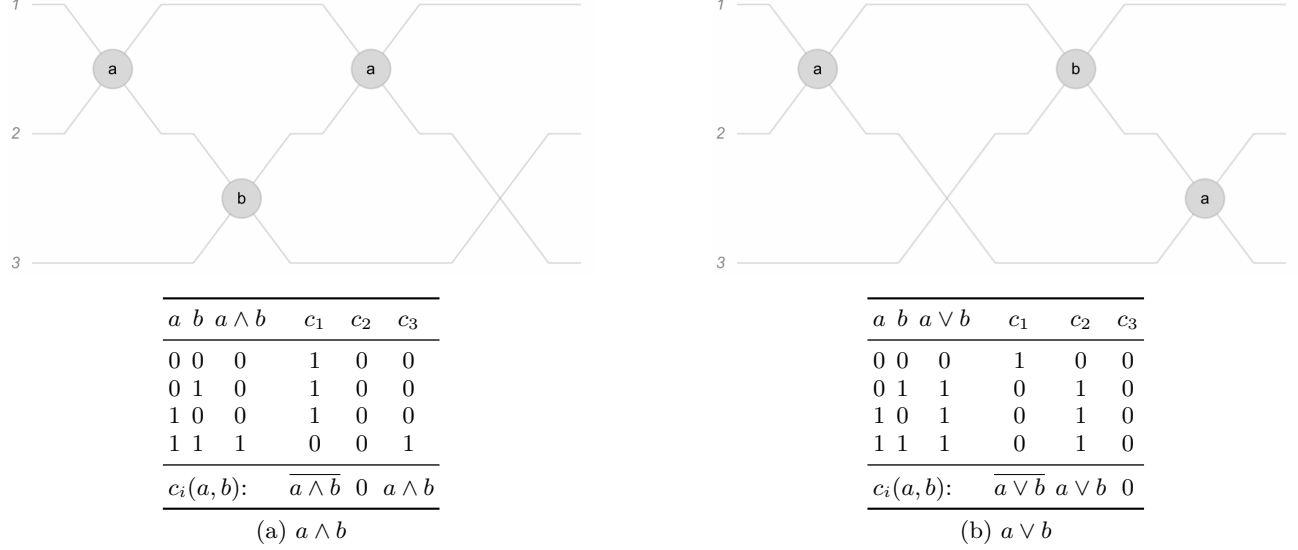


FIG. 9. DL circuits for binary logic operations. (a) $a \wedge b$ and (b) $a \vee b$, showing the graphical realizations (top) and the corresponding outputs $c_i(a, b)$ (bottom). The figure is equivalent to Fig. 2, taking into account that Fig. 2 presents the generally scalable configuration, whereas here its instantiation is given in the form of a 1-gate circuit.

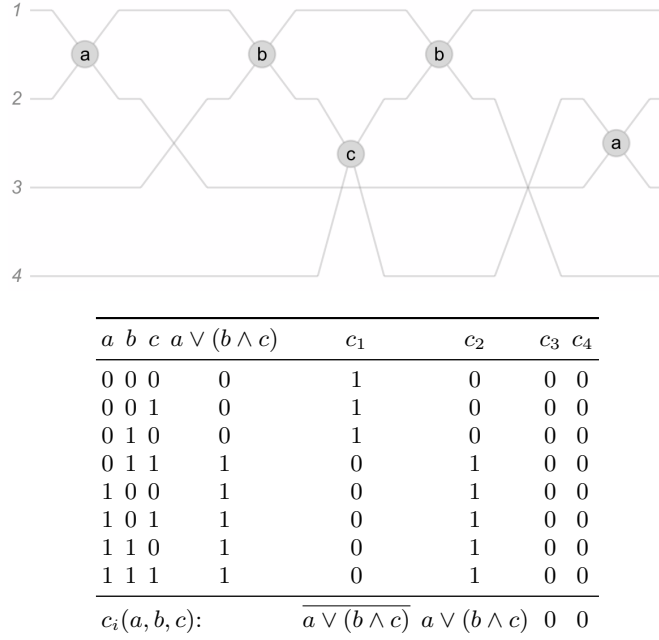
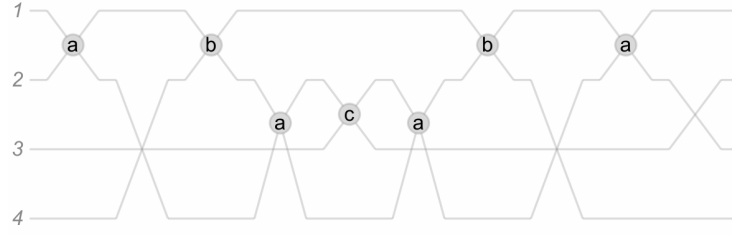
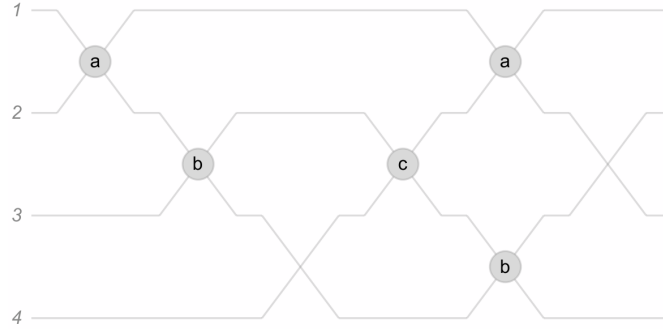


FIG. 10. Direct circuit for $a \vee (b \wedge c)$: (top) graphical realization and (bottom) the corresponding outputs $c_i(a, b, c)$.

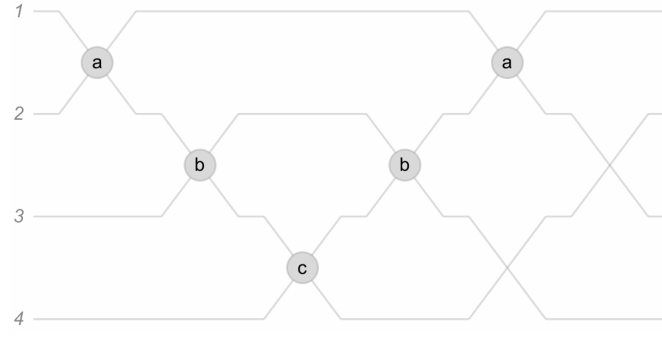


a	b	c	$(a \vee b) \wedge c$	c_1	c_2	c_3	c_4	c_5
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	1	0	1	0	0	0
1	0	0	0	1	0	0	0	0
1	0	1	1	0	1	0	0	0
1	1	0	0	1	0	0	0	0
1	1	1	1	0	1	0	0	0
$c_i(a, b, c):$				$\overline{(a \vee b) \wedge c}$	$(a \vee b) \wedge c$	0	0	0

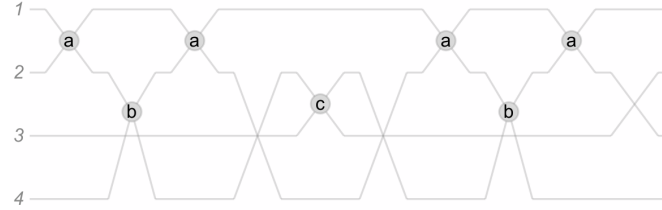
FIG. 11. Direct circuit for $(a \vee b) \wedge c$ and the corresponding outputs $c_i(a, b, c)$.

a	b	c	$a \wedge (b \vee c)$	c_1	c_2	c_3	c_4
0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0
0	1	0	0	1	0	0	0
0	1	1	0	1	0	0	0
1	0	0	0	1	0	0	0
1	0	1	1	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	1	0	1	0	0
$c_i(a, b, c):$				$\overline{a \wedge (b \vee c)}$	$a \wedge (b \vee c)$	0	0

FIG. 12. Direct circuit for $a \wedge (b \vee c)$ and the corresponding outputs $c_i(a, b, c)$.

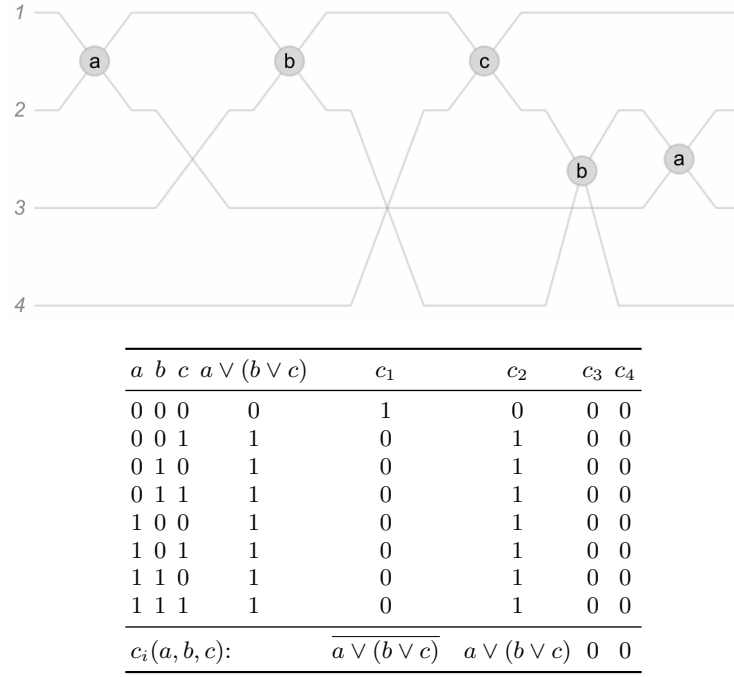
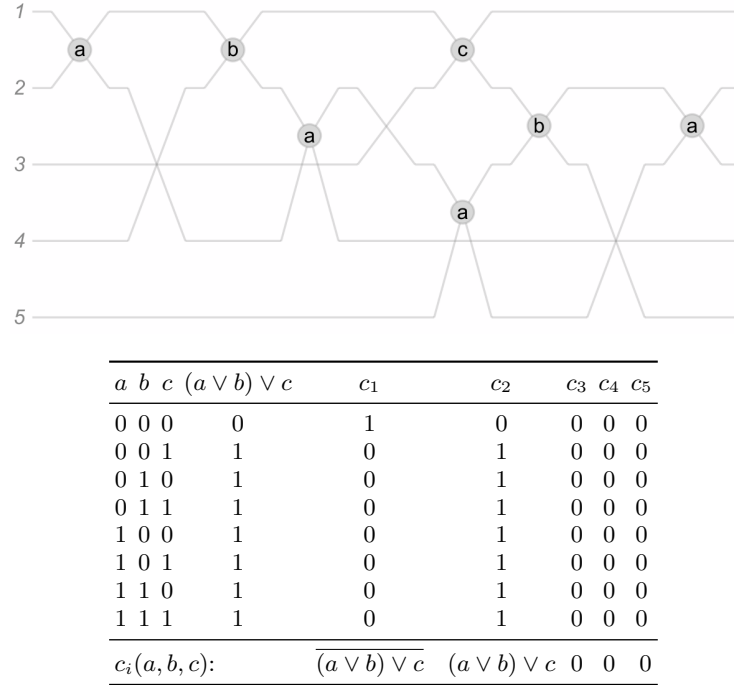


a	b	c	$a \wedge (b \wedge c)$	c_1	c_2	c_3	c_4
0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0
0	1	0	0	1	0	0	0
0	1	1	0	1	0	0	0
1	0	0	0	1	0	0	0
1	0	1	0	1	0	0	0
1	1	0	0	1	0	0	0
1	1	1	1	0	1	0	0
$c_i(a, b, c):$				$\overline{a \wedge (b \wedge c)}$	$a \wedge (b \wedge c)$	0	0

FIG. 13. Direct circuit for $a \wedge (b \wedge c)$ and the corresponding outputs $c_i(a, b, c)$.

a	b	c	$(a \wedge b) \wedge c$	c_1	c_2	c_3	c_4	c_5
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	1	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0
1	0	1	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0
1	1	1	1	0	1	0	0	0
$c_i(a, b, c):$				$\overline{(a \wedge b) \wedge c}$	$(a \wedge b) \wedge c$	0	0	0

FIG. 14. Attempt to construct the direct circuit for $(a \wedge b) \wedge c$ and the corresponding outputs $c_i(a, b, c)$. This configuration is the only one for which the truth table does not correspond to the intended composition.

FIG. 15. Direct circuit for $a \vee (b \vee c)$ and the corresponding outputs $c_i(a, b, c)$.FIG. 16. Direct circuit for $(a \vee b) \vee c$ and the corresponding outputs $c_i(a, b, c)$.

FUNDING

This work was funded by Germany's Excellence Strategy within the Cluster of Excellence PhoenixD (EXC 2122, Project ID 390833453).

DATA AVAILABILITY

All data supporting the findings of this study are available within the paper and its Supplementary Information.

CODE AVAILABILITY

The Mathematica package implementing the Directed Logic framework and all algorithms described in this work is publicly available at GitHub: <https://github.com/ihar-babushkin/DirectedLogic>. The repository also contains the notebook used to reproduce the figures and truth tables presented in this paper.

* babushkin@iqo.uni-hannover.de

- [1] J. Hardy and J. Shamir, Optics inspired logic architecture, *Opt. Express* **15**, 150 (2007).
- [2] W. Bogaerts, D. Pérez, J. Capmany, D. A. B. Miller, J. Poon, D. Englund, F. Morichetti, and A. Melloni, Programmable photonic circuits, *Nature* **586**, 207 (2020).
- [3] Z. Ying, C. Feng, Z. Zhao, S. Dhar, H. Dalir, J. Gu, Y. Cheng, R. Soref, D. Z. Pan, and R. T. Chen, Electronic-photonic arithmetic logic unit for high-speed computing, *Nature communications* **11**, 1 (2020).
- [4] R. Tang, R. Tanomura, T. Tanemura, and Y. Nakano, Ten-port unitary optical processor on a silicon photonic chip, *ACS Photonics* **8**, 2074 (2021).
- [5] H. Zhou, J. Dong, J. Cheng, W. Dong, C. Huang, Y. Shen, Q. Zhang, M. Gu, C. Qian, H. Chen, *et al.*, Photonic matrix multiplication lights up photonic accelerator and beyond, *Light: Science & Applications* **11**, 1 (2022).
- [6] S. Ning, H. Zhu, C. Feng, J. Gu, Z. Jiang, Z. Ying, J. Midkiff, S. Jain, M. H. Hlaing, D. Z. Pan, and R. T. Chen, Photonic-electronic integrated circuits for high-performance computing and ai accelerators, *Journal of Lightwave Technology* **42**, 7834 (2024).
- [7] D. Pérez-López and L. Torrijos-Morán, Large-scale photonic processors and their applications, *Nature Reviews Electrical Engineering* 10.1038/s44310-025-00075-4 (2025).
- [8] I. Bente, S. Taheriniya, F. Lenzini, F. Brücknerhoff-Plückelmann, M. Kues, H. Bhaskaran, C. D. Wright, and W. H. P. Pernice, The potential of multidimensional photonic computing, *Nature Reviews Physics* **7**, 439 (2025).
- [9] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin, *et al.*, Single-chip microprocessor that communicates directly using light, *Nature* **528**, 534 (2015).
- [10] A. H. Atabaki, S. Moazeni, F. Pavanello, H. Gevorgyan, J. Notaros, L. Alloatti, M. T. Wade, C. Sun, S. A. Kruger, H. Meng, *et al.*, Integrating photonics with silicon nanoelectronics for the next generation of systems on a chip, *Nature* **556**, 349 (2018).
- [11] I. Babushkin, O. Melchert, U. Morgner, and A. Demircan, Photon trapping in a time cavity flying with a speed of light, *Nature Communications* 10.21203/rs.3.rs-1982423/v1 (2026), accepted for publication.
- [12] A. Demircan, A. B. Evlyukhin, U. Morgner, O. Melchert, I. Babushkin, S. Zhang, F. Biancalana, C. Mei, J. Fan, P. Yang, G. Steinmeyer, P. Lalanne, O. D. Miller, H. Barati Sedeh, N. M. Litchinitser, V. Vaičaitis, O. Balachninaite, O. Takayama, R. Malureanu, A. V. Lavrinenko, B. Wetzl, B. P. Chaves, A. Tonello, V. Couderc, V. R. Tuz, A. Husakou, L. Bergé, X. Davoine, L. Gremillet, J. Vábe, F. Catoire, S. Skupin, P. Polynkin, J. Barker, J. Palastro, S. Sergeyev, H. Kbashi, S. K. Turitsyn, E. Menshikov, P. Franceschini, C. De Angelis, M. V. Gorkunov, Y. S. Kivshar, S. Beuchler, B. Endtmayer, U. Langer, A. Schafelner, T. Wick, F. Binkowski, F. Betz, M. Hammerschmidt, P.-I. Schneider, L. Zschiedrich, S. Burger, Y. Mahlau, L. Berg, M. Lindauer, B. Rosenhahn, P. Garg, V. Asadchy, E. Galiffi, S. Horsley, R. Sapienza, C. Rockstuhl, A. Karabchevsky, Y. Schrödel, H. Malang, S. Wismath, S. Soennecken, J. Helge Dörsam, T. Lang, M. Kupnik, H. Tuennermann, C. M. Heyl, B. Stiller, L. Zheng, A. Günther, W. Kowalsky, B. Roth, J. Duc Tran, V. E. Babicheva, S. Chabi, A. Crandall Jones, H. Kim, E. H. Lock, K. Sun, Z. Han, M. Hinkelmann, A. Wienne, M. Kues, D. Kracht, A. Calà Lesina, M. Fröhlich, L. Biesterfeld, L. Klepzig, D. A. Rudolph, O. Strolka, A. Niebur, J. Unseld, L. Kahlmeyer, A. Antanovich, J. Lauth, L. Fütterer, L. Overmeyer, E. V. Schlein, V. S. Vasylovskiy, B. N. Chichkov, J. Wohletz, M. Falkner, U. Wüllner, T. Biermann, R. Lachmayer, P. Rübeling, M. Kues, B. A. Malomed, L. Wegener, P. J. Walla, S. Athanasiou, O. J. F. Martin, P. Krause, and A. Bande, Roadmap on 5d photonics for nano and beyond, *Nano Futures* (2026).
- [13] A. Demircan, S. Amiranashvili, and G. Steinmeyer, Controlling light by light with an optical event horizon, *Phys. Rev. Lett.* **106**, 163901 (2011).
- [14] C. Husko, T. D. Vo, B. Corcoran, J. Li, T. F. Krauss, and B. J. Eggleton, Ultracompact all-optical xor logic gate in a slow-light silicon photonic crystal waveguide, *Opt. Express* **19**, 20681 (2011).
- [15] N. K. Langford, S. Ramelow, R. Prevedel, W. J. Munro, G. J. Milburn, and A. Zeilinger, Efficient quantum computing using coherent photon conversion, *Nature* **478**, 360 (2011).
- [16] V. Jandieri, R. Khomeriki, T. Onoprishvili, D. H. Werner, J. Berakdar, and D. Erni, Functional all-optical logic gates for true time-domain signal processing in nonlinear photonic crystal waveguides, *Opt. Express* **28**, 18317 (2020).
- [17] R. Yanagimoto, E. Ng, M. Jankowski, H. Mabuchi, and R. Hamerly, Temporal trapping: a route to strong coupling and deterministic optical quantum computation, *Optica* **9**, 1289 (2022).

- [18] I. Babushkin, A. Demircan, M. Kues, and U. Morgner, Wave-shape-tolerant photonic quantum gates, *Phys. Rev. Lett.* **128**, 090502 (2022).
- [19] Z. Li, J. Chen, Y. Sua, Z. Ma, C. Tang, and Y.-P. Huang, Invertible all-optical logic gate on chip, *Opt. Lett.* **49**, 3388 (2024).
- [20] R. Yanagimoto, B. A. Ash, M. M. Sohoni, M. M. Stein, Y. Zhao, F. Presutti, M. Jankowski, L. G. Wright, T. Onodera, and P. L. McMahon, Programmable on-chip nonlinear photonics, *Nature* **649**, 330 (2026).
- [21] M. Dai, R. Yang, W. Yu, P. Cai, and X. Wang, Ultrafast low-loss optical logic gates: Recent advances in mechanisms, materials, structures and performance toward all-optical computing, *Laser & Photonics Reviews* **n/a**, e03161 (2026).
- [22] H. J. Caulfield and S. Dolev, Why future supercomputing requires optics, *Nature Photonics* **4**, 261 (2010).
- [23] J. Shamir, H. J. Caulfield, W. Micelli, and R. J. Seymour, Optical computing and the fredkin gates, *Appl. Opt.* **25**, 1604 (1986).
- [24] H. Caulfield, A. Zavalin, and L. Qian, Zero-energy optical logic: can it be practical?, *The Journal of Supercomputing* , 1 (2011), 10.1007/s11227-011-0583-7.
- [25] Y. Tian, L. Zhang, R. Ji, L. Yang, P. Zhou, H. Chen, J. Ding, W. Zhu, Y. Lu, L. Jia, Q. Fang, and M. Yu, Proof of concept of directed or/nor and and/nand logic circuit consisting of two parallel microring resonators, *Opt. Lett.* **36**, 1650 (2011).
- [26] J. Ding, L. Yang, Q. Chen, L. Zhang, and P. Zhou, Demonstration of a directed xnor/xor optical logic circuit based on silicon mach-zehnder interferometer, *Optics Communications* **395**, 183 (2017).
- [27] H. Xiao, D. Li, Z. Liu, X. Han, W. Chen, T. Zhao, Y. Tian, and J. Yang, Experimental realization of a cmos-compatible optical directed priority encoder using cascaded micro-ring resonators, *Nanophotonics* **7**, 727 (2018).
- [28] C. Feng, Z. Ying, Z. Zhao, J. Gu, D. Z. Pan, and R. T. Chen, Toward high-speed and energy-efficient computing: A wdm-based scalable on-chip silicon integrated optical comparator, *Laser & Photonics Reviews* **15**, 2000275 (2021).
- [29] H. Xiao, X. Han, Y. Jiang, G. Ren, A. Mitchell, D. Gao, J. Yang, and Y. Tian, Demonstration of various optical directed logic operations by using an integrated photonic circuit, *Optics Letters* **46**, 2457 (2021).
- [30] S. Sharma and S. Roy, A survey on design and synthesis techniques for photonic integrated circuits, *The Journal of Supercomputing* **77**, 4332 (2021).
- [31] C. Qiu, H. Xiao, L. Wang, and Y. Tian, Recent advances in integrated optical directed logic operations for high performance optical computing: a review, *Frontiers of Optoelectronics* **15**, 1 (2022).
- [32] M. Yuan, Y. Li, H. Xiao, X. Zhou, P. Cao, L. Cheng, G. Ren, Q. Hao, J. Xue, A. Mitchell, and Y. Tian, Reconfigurable optical directed logic circuits based on mode division multiplexing technology, *IEEE Photonics Journal* **15**, 1 (2023).
- [33] M. Alkhazragi *et al.*, Reconfigurable optical logic in silicon platform, *Scientific Reports* **14**, 10.1038/s41598-024-56463-x (2024).
- [34] W. Wang, X. Yu, S. Lai, H. Wu, S. Yan, and M. Tang, Optically-reconfigurable integrated optical directed logic computing based on silicon photonics, *Optics Express* **32**, 24007 (2024).
- [35] Z. Xu, B. Tang, X. Zhang, J. F. Leong, J. Pan, S. Hooda, E. Zamburg, and A. V.-Y. Thean, Reconfigurable nonlinear photonic activation function for photonic neural network based on non-volatile opto-resistive ram switch, *Light: Science & Applications* **11**, 1 (2022).
- [36] H. J. Caulfield, R. A. Soref, L. Qian, A. Zavalin, and J. Hardy, Generalized optical logic elements—goles, *Optics Communications* **271**, 365 (2007).
- [37] D. A. Miller, The role of optics in computing, *Nature Photonics* **4**, 406 (2010).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [supplement.pdf](#)