scientific reports

OPEN

Machine learning optimal ordering in global routing problems in semiconductors

Heejin Choi[1,3], Minji Lee[1,3], Chang Hyeong Lee[1], Jaeho Yang[2] & Rak-Kyeong Seong[1]✉

In this work, we propose a new method for ordering nets during the process of layer assignment in global routing problems. The global routing problems that we focus on in this work are based on routing problems that occur in the design of substrates in multilayered semiconductor packages. The proposed new method is based on machine learning techniques and we show that the proposed method supersedes conventional net ordering techniques based on heuristic score functions. We perform global routing experiments in multilayered semiconductor package environments in order to illustrate that the routing order based on our new proposed technique outperforms previous methods based on heuristics. Our approach of using machine learning for global routing targets specifically the net ordering step which we show in this work can be significantly improved by deep learning.

Routing in semiconductor chip and package design is the process of creating and designing electrical connections between different components of a chip or package, such as transistors, gates, input and output pins, solder balls and other components depending on the design specifications of the chip and package. For example, for a Fine Pitch Ball Grid Array (FBGA) package with up to approximately 500 pins, an expert human designer may take up to 48 hours to complete a routing design for the semiconductor package. This does not take into account design testing for the design candidate which might end up sub-optimal for manufacturing. As a result, design automation and optimization in semiconductor chip and package design has been playing a significant role for the last few decades[1–6]in order to simplify the task of routing design in semiconductor chip and packages. This field of research is also known today as electronic design automation (EDA)[7–9] in computer-aided design (CAD).

Automating and optimizing the routing process is a quintessential problem in EDA for semiconductor chips and packages. This is because the design complexity that results from a given routing solution directly impacts the performance, power consumption, signal integrity, manufacturability and costs of producing a semiconductor chip and package[9,10]. Moreover, with an increasing number of components to be connected by non-intersecting electrical connections in a 3-dimensional chip and package environment, the routing problem becomes exponentially more difficult not just to automate, but also to optimize against design and performance constraints.

With this exponentially increasing difficulty in mind, a series of innovations[11]in EDA introduced the concept of global routing[12–17] with the aim of simplifying and isolating the problem of routing from the rest of the design problem for semiconductor chips and packages. The main idea is to separate the routing problem into two parts: *global routing* and *detailed routing*[18–21]. In global routing, the locations of areas that need to be connected during the routing process are embedded into a grid graph which discretizes and simplifies the multi-layered routing environment. After connections are obtained in global routing, the grid is mapped back during detailed routing to the original routing environment by implementing design constraints such as wire-widths and wire separations.

In this work, we concentrate on the process of global routing in multi-layered routing problems, especially in semiconductor package substrate design. In particular, we focus on the layer assignment problem in global routing[2,22–24]. The layer assignment problem occurs when a multilayered routing problem is simplified as a routing problem in a 2-dimensional grid. When the routing solution is found in the 2-dimensional grid, the routing solution is mapped to a k-layered grid graph by assigning connections between layers and layer numbers to the original 2-dimensional routing solution. The corresponding algorithm was first introduced in[2]. There it was noted that if the routing problem consisted of several nets, the layer assignment of the 2-dimensional routing solutions for each individual net heavily depended on the order in which each net solution was assigned layers during the layer assignment process.

[1]Department of Mathematical Sciences, Ulsan National Institute of Science and Technology, Ulsan, South Korea. [2]Samsung SDS, AI Advanced Research Lab, Samsung R&D Campus, Seocho-Gu, Seoul, South Korea. [3]These authors contributed equally: Heejin Choi and Minji Lee. ✉email: seong@unist.ac.kr

A solution to the ordering optimization problem proposed in the original work in[2]was to assign each net and its 2-dimensional routing solution a heuristic score according to which a net order can be computed. This heuristic score function depended on features such as the total routing length, overflow and vertices used in the 2-dimensional routing solution. In this work, we propose a new method of ordering nets and their 2-dimensional routing solution for the layer assignment process in global routing. Our method is based on machine learning a collection of features of the 2-dimensional routing solutions and to predict the most optimal net ordering that results in the most desired 3-dimensional routing solution after layer assignment. Our proposed new method supersedes and outperforms the heuristics-based net ordering method in[2] based on experimental results that we present in the following work.

Problem formulation

Multilayered routing is a critical problem in very large scale integration (VLSI) technology in semiconductor chip and package design[7–9]. Here, routing is conducted in two phases known as global routing[2,22–24]and detailed routing[25,26]. In the following work, we concentrate on global routing which can be conducted in two different approaches that are both illustrated in Figure 1. The first approach is to directly find the routing in the multilayered routing environment, where the objects to be connected are distributed in different layers effectively requiring a routing in 3-dimensional space. Even though this approach outputs directly a 3-dimensional multilayered routing result with the benefit that the cost of connecting different layers can be directly taken into account in the 3-dimensional routing process, this approach is computational expensive becoming exponentially harder to solve with more layers and more nets in the problem. The second approach is to compress the multilayered routing environment into a single layer grid on which one can solve the routing problem first and then assign to each section of the routing a layer in order to get the 3-dimensional multilayered routing solution. Here, the problem becomes much more computational solvable, however it introduces a new decision step known as net ordering which is the main topic of discussion in this work.

In the following, we give a brief summary of the global routing process using the second approach with layer compression and layer assignment, and give a brief overview of the optimization problem involved in the net ordering.

Global routing algorithm based on layer compression

Global routing in a k-layer routing environment involves the following algorithmic steps:

1. Grid Graph: We obtain the k-layer grid graph $G^k = (V^k, E^k)$ from the 3-dimensional routing environment and routing problem. The grid vertices $v_i \in V^k$ correspond to grid regions in the k-th layer of the routing environment. Physical objects such as solder balls and metal pins on the routing environment that need to be connected are represented by pins p_m^k, which are each associated to one of the grid vertices $v_i \in V^k$. The boundary separating two neighboring grid regions is mapped to an edge $e_{ij}^k = (v_i^k, v_j^k)$ in G^k. Next to *boundary edges* E_b^k that connect neighboring grid vertices on the same layer, we also have *via edges* E_v^k that

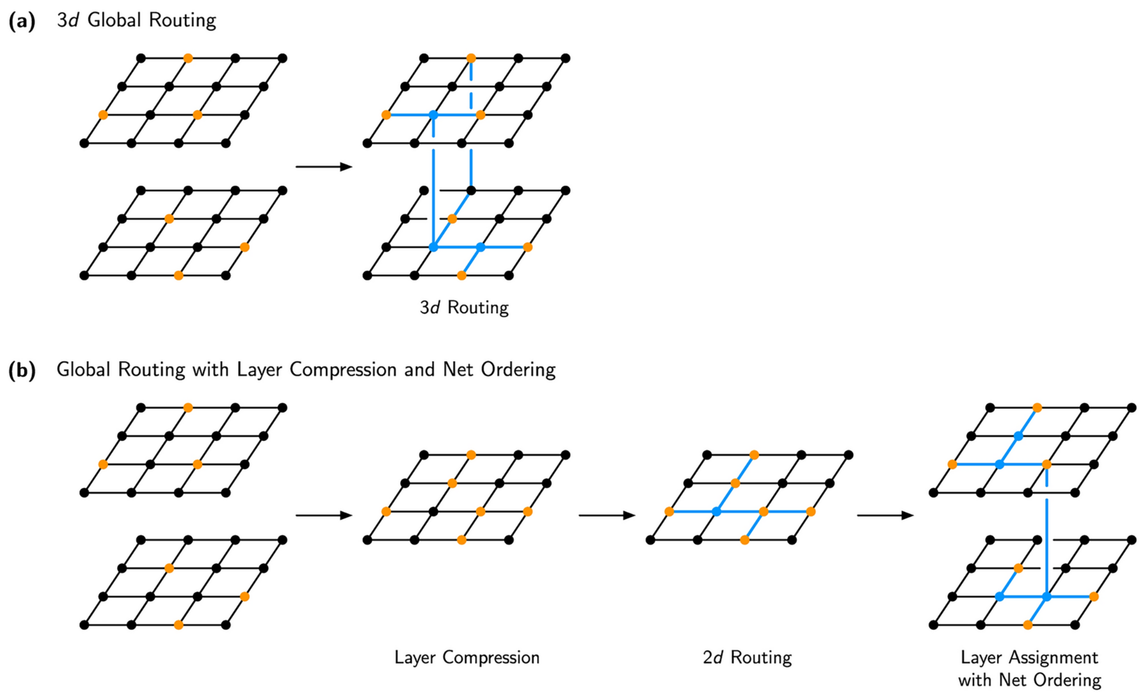

Fig. 1. Comparison between (**a**) 3-dimensional global routing directly inside the multilayered environment, and (**b**) global routing with layer compression and net ordering.

connect neighboring grid vertices in adjacent layers. An edge e_{ij}^k is assigned a *capacity* $c(e_{ij}^k) \in \mathbb{Z}^+$, which is the number of available connections that are possible, and the *length* $l(e_{ij}^k) \in \mathbb{R}^+$, which the physical wirelength in the actual routing environment for any connection that passes through e_{ij}^k. In this work, we will assume that all edge wirelengths are the same such that for all $e_{ij}^k, e_{uv}^k \in E^k, l(e_{ij}^k) = l(e_{uv}^k)$. Figure 2 shows an example of a routing environment divided into 2×2 grid regions that correspond to vertices $v_1^k, \ldots, v_4^k$ in a grid graph.

2. Routing Problem and Nets: Vertices $v_i \in V^k$ belong to nets $n_m^k \in N^k$, and vertices corresponding to the same net have to be connected during the routing process. Given a net $n_m^k \in N^k$, a *global routing solution* on layer *k* of the routing environment is a connected subgraph $T_m^k = (V_m^k, E_m^k)$ of the grid graph G^k, where $V_m^k \subset V^k$ and $E_m^k \subset E^k$ are respectively the subset of vertices and edges that contribute to the global routing solution for n_m^k. We note that for a given net n_m^k, there can be more than one routing solution T_m^k. If every net n_m^k in the global routing problem is assigned a routing solution T_m^k, the collection $S^k = \{T_m^k\}$ is known as the *complete global routing solution* for (G^k, N^k).
3. Layer Compression: 1-layer compression takes the *k*-layer grid graph $G^k = (V^k, E^k)$ and maps it to $G^1 = (V^1, E^1)$. Figure 4 illustrates the 1-layer compression for a 3-layer grid graph. This reduction into a 2-dimensional routing problem still encapsulated the constraints given in the 3-dimensional routing problem. These preserved constraints include the *compressed capacity*, which is $c(e_i^1) = \sum_{e_m^k \in ES(e_i^1)} c(e_m^k)$.
4. $2d$ Global Routing : Solve the 2-dimensional routing problem by finding the 1-layer routing solution $S^1 = \{T_m^1\}$. Solutions are found using algorithms such as the *Kruskal's Algorithm (KA)*[27], based on the search algorithm for the *minimum spanning tree* along the grid graph G^1, and the *Steiner Tree Algorithm (ST)*[28] based on an algorithm for identifying the shortest possible tree formed by a subset of edges in G^1.
5. Net Ordering: Every net in the 1-layer compression is connected by the 2*d* routing result. In order to now assign layers to each routing segment in T_m^1, we have to first identify in which *order* nets are assigned layer information. Depending on the net ordering, the 3-dimensional global routing result will become different. This process is known as *net ordering* and is illustrated in Figure 3.
6. Layer Assignment: The 2-dimensional routing solution $S^1 = \{T_m^1\}$ is lifted to a solution on the *k*-layer grid $S^k = \{T_m^k\}$ by a process called *layer assignment*.

In the following work, we will put a particular emphasis on the last two steps of the global routing process known as *net ordering* and *layer assignment*.

Net ordering

The order in which subgraphs T_m^1 for each net n_m^1 are lifted to 3 dimensions affects dramatically the overall 3*d* global routing solution on G^k. This is because given a certain finite non-zero capacity $c(e_{ij}^k) \in \mathbb{Z}^+$ on all edges $e_{ij}^k \in E^k$ in the 3-dimensional grid graph G^k, the 3*d* global routing solution T_m^k after layer assignment should not contain a grid graph G^k with edges that have a *overflow* given by,

$$o(e_{ij}^k) = \begin{cases} d(e_{ij}^k) - c(e_{ij}^k) & \text{if } d(e_{ij}^k) > c(e_{ij}^k) \\ 0 & \text{otherwise} \end{cases} , \tag{1}$$

where $d(e_{ij}^k)$ is the number of subgraphs T_m^k containing edge e_{ij}^k, i.e. the *demand* for edge e_{ij}^k, and $c(e_{ij}^k)$ is the capacity of edge e_{ij}^k. Depending on which subgraph T_m^1 is lifted first to T_m^k, the distribution of demand in the *k*-layer graph G^k changes significantly. With the design constraint that overflow is kept at $o(e_{ij}^k) = 0$ for all edges $e_{ij}^k \in E^k$, the distribution of demand $d(e_{ij}^k)$ determined by the ordering of T_m^1 heavily affects the overall *k*-layer global routing result S^k.

As a result, net ordering significantly affects the overall 3-dimensional global routing solution S^k. We define a *net ordering* as a ordered sequence,

$$(T_{m_1}^1, T_{m_2}^1, \ldots, T_{m_{N_{\text{nets}}}}^1) \ : \ \text{score}(T_{m_1}^1) \geq \text{score}(T_{m_2}^1) \geq \cdots \geq \text{score}(T_{m_{N_{\text{nets}}}}^1) \ , \tag{2}$$

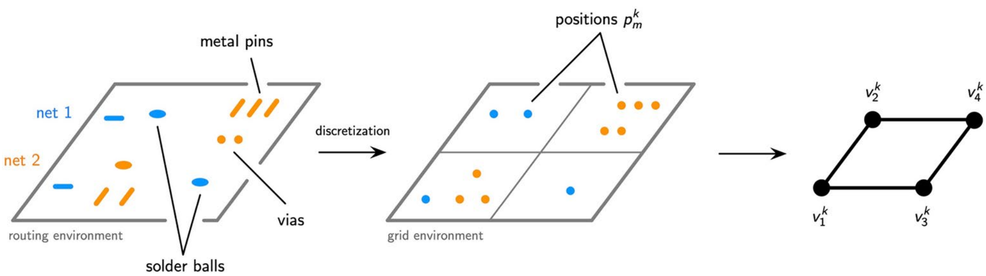

Fig. 2. Process of Discretization during Global Routing, where physical objects in the routing environment are interpreted as vertices in a grid graph G^k.

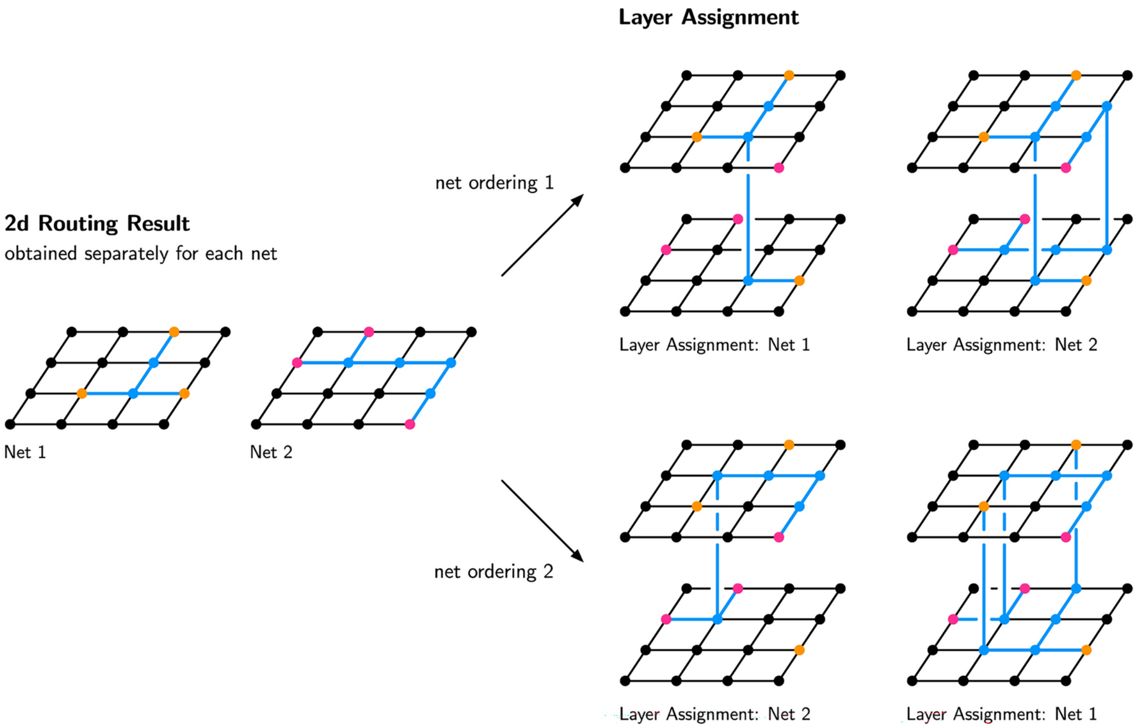

Fig. 3. Net ordering during layer assignment of the $2d$ routing results affects the overall $3d$ routing result in global routing.

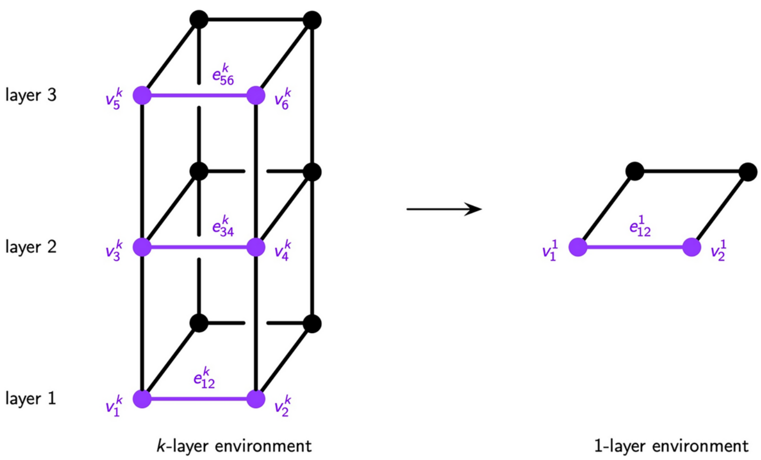

Fig. 4. 1-layer compression for a 3-layer grid graph. Under the compression, equivalent vertices v_m^k and edges e_{ij}^k are mapped to single vertices v_m^1 and edges e_{ij}^1 in the 1-layer compressed graph G^1.

where N_{nets} is the number of nets in N^1, and $\text{score}(T_m^1) \in \mathbb{R}_{\geq 0}$ is a score value assigned to each 1-layer routing solution T_m^1 corresponding to net n_m^1, which is used for the net ordering. In[2], the score function was defined heuristically for $T_m^1 = (V_m^1, E_m^1)$ as follows

$$\text{score}(T_m^1) = \frac{\alpha}{l(T_m^1)} + \beta N_{\text{pins}}(T_m^1) + \gamma \bar{\rho}(T_m^1) \ , \tag{3}$$

where $l(T_m^1)$ is the total wirelength for the subgraph T_m^1 defined by $l(T_m^1) = \sum_{e_{ij}^1 \in E_m^1} l(e_{ij}^1)$. $N_{\text{pins}}(T_m^1) = |P_m^1|$ is the number of pins in P_m^1 of the subgraph T_m^1 corresponding to net n_m^1. Furthermore, the last term in (3) depends on the average net density $\rho(T_m^1)$ for the subgraph T_m^1, which is defined as follows

$$\overline{\rho}(T_m^1) = \frac{\sum_{e_{ij}^1 \in E_m^1} d(e_{ij}^1)}{\sum_{e_{ij}^1 \in E_m^1} c(e_{ij}^1)} \ , \tag{4}$$

where $d(e_{ij}^1)$ is the demand on the edge e_{ij}^1 in the 1-layer routing solution S^1 and $c(e_{ij}^1)$ is the capacity of the edge e_{ij}^1. According to[2], the coefficients α, β and γ are determined by the designed constraints of the 3*d* routing problem.

We will see in the following work that a net ordering algorithm based on machine learning outperforms the heuristic function in (3) in various experiments.

Layer assignment

We review here briefly an algorithm known as *single-net optimal layer assignment (SOLA)* based on[2]. We start with an *k*-layer grid G^k where all edges of the *k*-layer grid have demand $d(e_{ij}^k) = 0$ and overflow $o(e_{ij}^k) = 0$. Through layer assignment of a compressed 2*d* global routing solution $S^1 = \{T_m^1\}$ with net ordering $(T_{m_1}^1, T_{m_2}^1, \ldots, T_{m_{N_{\text{nets}}}}^1)$, we go sequentially through the subgraphs T_m^1 and layer assign their edges into G^k. This process updates $d(e_{ij}^k)$ with the constraint that $o(e_{ij}^k) = 0$ has to be kept for all edges.

Given the minimum spanning tree $T_m^1 = (V_m^1, E_m^1)$, the edges in E_m^1 can be assigned directions such that one of the pins p_{ij}^{source} in P_m^1 is the source where all edges begin from, and all other pins p_{ij}^{sink} in P_m^1 are the sinks where all edges end. This orientation of edges in E_i^1 is illustrated in a simple example in Figure 5. Under such an orientation of edges, every vertex in $v_i^1 \in V_m^1$ has a natural set of children vertices defined as

$$ch(v_i^1) = \{w \mid \vec{e} = (v_i^1, w) \ , \ \vec{e} \in E_m^1\} \ , \tag{5}$$

where $\bar{e}$ is an oriented edge starting from vertex v_i^1 and ending at vertex *w*. We can also define the parent vertex for v_i^1 as

$$par(v_i^1) = w \ , \ \vec{e} = (w, v_i^1) \in E_m^1 \ , \tag{6}$$

where $\bar{e}$ is an oriented edge starting from vertex *w* and ending at vertex v_i^1. In this case, we call $\vec{e} = (w, v_i^1)$ also the parent edge $par_e(v_i^1)$

We now take the 1-layered 2*d* routing solution to be positioned at the $k = 1$ layer of the *k*-layer grid graph G^k. Using the orientation of edges for a given subtree T_m^1, we can now define a recursive heuristic cost function which measures the cost of moving a parent edge $par_e(v_i^1)$ for a vertex v_i^1 to layer *r* of the *k*-layered graph G^k,

$$mvc(v_i^1, r) = \min_{\substack{1 \le r_1 \le k \\ \vdots \\ 1 \le r_q \le k}} \left(\sum_{j=1}^{q} mvc(v_j^1, r_j) + vc(v_i^1) \right) \ , \tag{7}$$

where $q = |ch(v_i^1)|$ is the number of children vertices of v_i^1 and $vc(v_i^1)$ is the cost of adding a via edge between the first layer and the *r*-th layer. We note that the sum in (7) is over all children vertices of v_i^1 labelled by $j = 1, \ldots, |ch(v_i^1)|$. The key of the SOLA algorithm is to minimize the overall cost $mvc(v_i^1, r)$ for all vertices v_i^1 in subtree solution T_m^1, with the constraint that all edge overflows are kept at zero, $o(e_{ij}^k) = 0$.

Using this recursive score function, every edge in a subgraph T_m^1 can be lifted to an optimal layer *r*. The addition of via edges that connect vertices between different layers, the SOLA algorithm ensures that the resulting T_m^k subgraph in G^k is a connected between layers. Figure 6 illustrates an example where a 1-layer routing solution S^1 is lifted to a *k*-layer routing solution S^k.

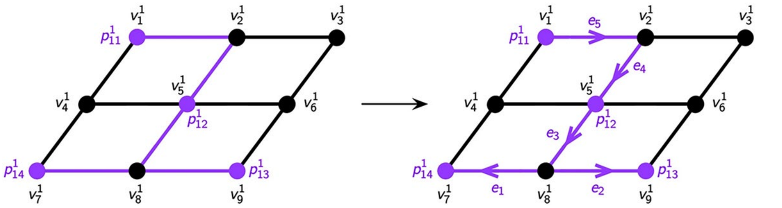

Fig. 5. An orientation is assigned to edges e_{ij} of a minimum spanning tree T_m^1 by choosing p_{11}^1 as the source and all other vertices as sinks of the orientation.

We make use of the SOLA algorithm to obtain for a given specific net ordering a *k*-layer routing solution $S^k = \{T_m^k\}$. In addition to SOLA, there is a complementary algorithm known as *Accurate and Predictable Examination for Congestion*(APEC)[2], which minimizes the overall overflow in the routing solution during layer assignment. Combined with SOLA, the two algorithms are known as *Congestion-Constrained Layer Assignment*(COLA)[2], which we use in our experiments in this work. By comparing different routing solutions $S^k(T^1_{m_1}, T^1_{m_2}, \ldots, T^1_{m_{N_{\text{nets}}}})$ for different net orderings $(T^1_{m_1}, T^1_{m_2}, \ldots, T^1_{m_{N_{\text{nets}}}})$, we can identify which net ordering results in the most optimal *k*-layer global routing solution. In the following sections, we present how we trained a deep neural network to predict the most optimal net ordering $(T^1_{m_1}, T^1_{m_2}, \ldots, T^1_{m_{N_{\text{nets}}}})$ that results in the most optimal *k*-layer global routing solution S^k. We show that our machine learning model outperforms the heuristics-based net ordering scoring function in (3) by conducting several experiments.

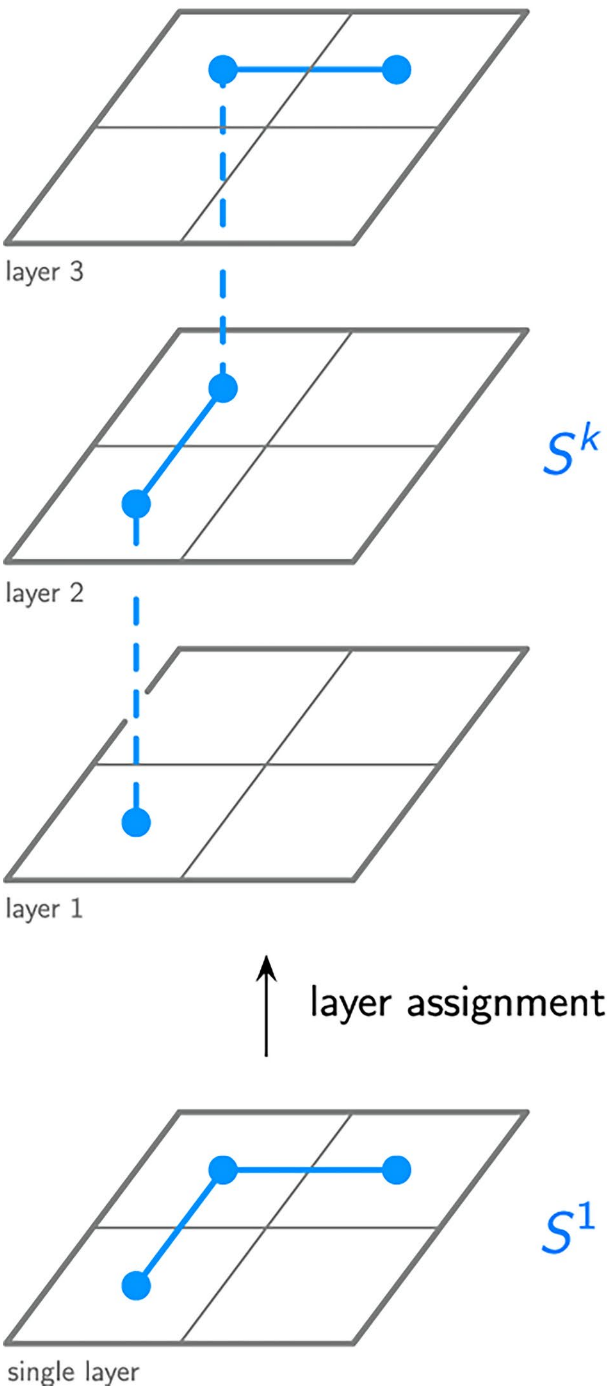

Fig. 6. Layer assignment of a 1-layer routing solution S^1 to a *k*-layer routing solution S^k.

Method

Features for machine learning

In this work, we present a machine learning model that takes as an input the collection of 1-layer $2d$ global routing solutions $S^1 = \{T_m^1\}$ and outputs the most optimal net ordering $(T_{m_1}^1, T_{m_2}^1, \ldots, T_{m_{N_{\text{nets}}}}^1)$ which results in the most optimal k-layer routing solution $S^k = \{T_m^k\}$. In order to train the deep neural network, we identify the following features from the 1-layer global routing solutions $T_m^1 = (V_m^1, E_m^1)$:

- Number of Pins $|P_m^k|$: This is the number of pins associated to vertices $v_i^k \in V^k$ and a given net n_m^k in a k-layer routing environment.
- Number of Pins $|P_m^1|$: This is the number of pins associated to compressed vertices $v_i^1 \in V^1$ and a given net n_m^1 in a single layer routing environment prior to layer assignment.
- Number of Vertices $|V_m^1|$: This is the number of vertices in V_m^1 in the $2d$ routing solution T_m^1 for a given net n_m^1.
- Overflow $o(E_m^1)$: This is the total overflow of the edges e_{ij}^m in E_m^1 in the $2d$ routing result T_m^1 for a given net n_m^1. The total overflow is given by $o(E_m^1) = \sum_{e_{ij}^m \in E_m^1} o(e_{ij}^k)$, where the overflow $o(e_{ij}^k)$ for an edge e_{ij}^k is defined in (1).
- Minimum Rectangle: We define a minimum rectangle that contains all of the vertices in the 2-dimensional routing solution S^1. This rectangle has the following dimensions

$$\Delta x_{min} = \max_{v_i^1 \in V^1} x(v_i^1) - \min_{v_j^1 \in V^1} x(v_j^1) \quad , \quad \Delta y_{min} = \max_{v_i^1 \in V^1} y(v_i^1) - \min_{v_j^1 \in V^1} y(v_j^1) \ , \tag{8}$$

where $x(v_i^1)$ and $y(v_i^1)$ are the x- and y-coordinates of the vertex $v_i^1 \in V^1$, respectively. The area of the rectangle is given by $A_{min} = \Delta x_{min} \cdot \Delta y_{min}$.

- Number of Branch Vertices $|V_{\text{branch}}^1|$: This is the number of vertices in the 2-dimensional routing solution S^1, which are branch points in T^1. A vertex $v_i^1 \in V^1$ is a branch point if $|ch(v_i^1)| > 1$ or $|par(v_i^1)| > 1$ .In summary, for a given k-layer routing solution S^k, we can obtain a feature vector of the following form,

$$\vec{f}(S^k) = \left(\{|P_m^k|\}, \{|P_m^1|\}, \{|V_m^1|\}, \{o(E_m^1)\}, \Delta x_{min}, \Delta y_{min}, A_{min}, |V_{\text{branch}}^1|\right) \ , \tag{9}$$

where the index $m = 1, \ldots, N_{\text{nets}}$ labels the nets in S^k.

Optimal net order

Our goal is to find the optimal net order based on criteria used in substrate routing to evaluate optimality in routing designs[29]. The criteria used in[29] to evaluate optimality uses features of the routing results S^k after layer assignment.

In our work, we simplify the optimality criteria in[29]. Given a $2d$ routing solution S^1, we use a candidate net ordering $no = (T_{m_1}^1, T_{m_2}^1, \ldots, T_{m_{N_{\text{nets}}}}^1)$ to obtain the corresponding layer assigned k-layer routing solution S^k in order to evaluate the optimality of the k-layer routing solution S^k as follows:

1. Given two k-layer routing solutions S_1^k and S_2^k corresponding to net orderings no_1 and no_2 associated to the same $2d$ routing solution S^1, we calculate the total overflow for S_1^k and S_2^k, which is given by $o(S^k) = \sum_{e_{ij}^k \in E^k} o(e_{ij}^k)$, where E^k is the set of edges in the layer assigned routing solution S^k, and $o(e_{ij}^k)$ is the overflow for edge $e_{ij} \in E^k$, which is defined in (1). The most optimal k-layer routing solution is determined by the one which has the smaller value for $o(S^k)$. If $o(S_1^k) = o(S_2^k)$, we calculate the maximum overflow for a routing solution instead. The maximum overflow $mo(S^k)$ for a routing solution S^k is defined as $mo(S^k) = \max_{e_{ij}^k \in E^k} o(e_{ij}^k)$. If for two routing solutions S_1^k and S_2^k we have $o(S_1^k) = o(S_2^k)$, then the most optimal k-layer routing solution S^k is determined by the one which has the smaller value for $mo(S^k)$.
2. If for two routing solutions S_1^k and S_2^k we have both $o(S_1^k) = o(S_2^k)$ and $mo(S_1^k) = mo(S_2^k)$, then we calculate the optimality score value which is defined as follows,

$$\text{Score}(S^k) = l(S^k) \times (1 + t_{run}) \ , \tag{10}$$

where L is the total wirelength of edges in S^k, $l(S^k) = \sum_{e_{ij}^k \in E^k} l(e_{ij}^k)$, and t_{run} is the processor runtime for layer assignment of S^1 to S^k. A smaller $\text{Score}(S^k)$ corresponds to a more optimal k-layer routing solution S^k. All experiments in this work are conducted on a server with Intel Xeon CPU E5-2687W.

Based on the above measures, we can assign an optimality value for each *k*-layer routing solution S^k that was obtained from a net ordering $no(S^k)$ for the layer assignment of S^1. For a single layer routing solution S^1 with N_{nets} number of nets, the total number of candidate net orderings is given by $N_{\text{ordering}} = (N_{\text{nets}})!$. The optimality of S^k takes an integer value in $[1, N_{\text{ordering}}]$ which represents the rank from most optimal, *i.e.* $\text{optimality}(S^k) = 1$, to least optimal, *i.e.* $\text{optimality}(S^k) = N_{\text{ordering}}$. The actual optimality value and ranking between two *k*-layer routing solutions S_1^k and S_2^k is decided by the following conditions based on the optimality measures discussed above,

$$\text{optimality}(S_1^k) < \text{optimality}(S_2^k) \Leftrightarrow \begin{cases} 1. & o(S_1^k) \le o(S_2^k) \\ 2. & mo(S_1^k) \le mo(S_2^k) \\ 3. & \text{Score}(S_1^k) < \text{Score}(S_2^k) \end{cases} . \tag{11}$$

Machine learning model

In[2], optimal net ordering was determined by a heuristic score function which we reviewed in (3). This static function assigns a score for each net n_m^1 in 2-dimensional routing solution S^1 and the net ordering is determined by the ordering of the score values. In our work, we replace the heuristic method in[2] with a deep learning model that orders the nets in S^1 in order to obtain the most optimal *k*-layer routing solution after layer assignment. Because our deep learning model is trained to result in the most optimal *k*-layer routing solution after layer assignment, it is qualitatively different to the heuristic ordering method in[2], which only depends on static features of the 1-layer routing solution S^1 without receiving any feedback from the *k*-layer routing solution S^k after layer assignment.

In order to train our deep-learning model, we generate for a given routing problem a 1-layer routing solution S^1 and construct from it the *k*-layer routing solutions S_h^k by using all possible net orderings no_h, where $h = 1, \ldots, N_{\text{ordering}}$ labels the individual net orderings. For each of these *k*-layer routing solutions S_h^k obtained by layer assignment of S^1 with a given net ordering no_h, we measure the optimality of S_h^k§. Furthermore, we extract the feature vector $\vec{f}(S_h^k)$ of the *k*-layer routing solutions S^k. Both the features vector $\vec{f}(S_h^k)$ and $\text{optimality}(S_h^k)$ are used to train our deep learning model.

The training data is segmented into groups. Each training data group corresponds to a given 1-layer routing solution S^1 and consists of all possible net orders labelled by $h = 1, \ldots, N_{\text{ordering}}$. By computing the *k*-layer routing solution S_h^k for each of the net orderings no_h, the net orders no_h and the corresponding solution S_h^k are ordered by determining $\text{optimality}(S_h^k)$. We also calculate for each solution S_h^k the features vector $\vec{f}(S_h^k)$. Our deep learning model takes for each S_h^k per data group as input the feature vector $\vec{f}(S_h^k)$ and outputs the optimality rank $\text{optimality}(S_h^k)$ for the net ordering no_h within the data group corresponding to S^1.

Although our training data is labelled by $\text{optimality}(S_h^k)$, the ranking done using $\text{optimality}(S_h^k)$ is within each data group corresponding to a single layer routing solution S^1, so that only the nets and net ordering no_h within the same group affect the ranking under $\text{optimality}(S_h^k)$. This means that even with the same feature values from both the 1-layer routing solution S^1 and *k*-layer routing solution S^k, the best net ordering under $\text{optimality}(S_h^k)$ can differ depending on the different net orderings no_h within the group corresponding to S^1. In order to address this issue, our deep learning model consists of two parts:

(1) Learning between Data Groups: The aim here is to train the deep learning model by using feature vectors $\vec{f}(S_h^k)$ across different data groups associated to different single layer routing solutions S^1.
(2) Learning within a Data Group: The aim here is to train the deep learning model by using feature vectors $\vec{f}(S_h^k)$ within a given data group associated to a single routing solution S^1.

Our deep learning model uses a multilayer perceptron (MLP) architecture. This is a fundamental and widely used deep learning model that makes use of a variety of activation and loss functions. Since the output of the last layer is the optimal net ordering *no* based on the ranking by $\text{optimality}(S_h^k)$, our deep learning model can be identified as a classification model. Accordingly, we use Mean Squared Error (MSE) and Cross-Entropy (CE), which are often used in classification models as loss functions. The MSE is defined as follows,

$$\text{Mean Squared Error (MSE)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 , \tag{12}$$

where *N* is the size of the labelled dataset, y_i is the expected output and $\hat{y}_i$ is the predicted output for the *i*-th datapoint. Similarly, the Cross-Entropy is defined as

$$\text{Cross-Entropy (CE)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C}\sum_{m=1}^{C} p_{ijm}\log(q_{ijm}) , \tag{13}$$

where in our deep learning models *N* is the number of data groups each corresponding to a single layer solution S_i^1, *C* corresponds to the number of nets N_{nets}, and p_{ij} is the probability N_{nets}-vector that indicates the actual optimal order *m* of the net n_j^k with a probability component $p_{ijm} = 1$, whereas all other components for the net n_j^k are taken to be 0. In comparison, q_{ij} is the predicted probability N_{nets}-vector which labels with a component q_{ijm} the predicted probability that the net n_j^k has order *m*. Note that the log-function in (13) is used to penalize more heavily incorrect probability predictions than correct ones.

Model	Layer 1 (Activation function)	Layer 2	Layer 3 (Activation function)	Loss function
Model 1	Linear (Hyperbolic Tangent)	Linear	Linear (Softmax)	MSE
Model 2	Linear (ReLU)	Linear	Linear (Softmax)	MSE
Model 3	Linear (Hyperbolic Tangent)	Linear	Linear (Hyperbolic Tangent)	Cross-Entropy

Table 1. Summary of the 3 different types of deep learning models used to predict the most optimal net ordering.

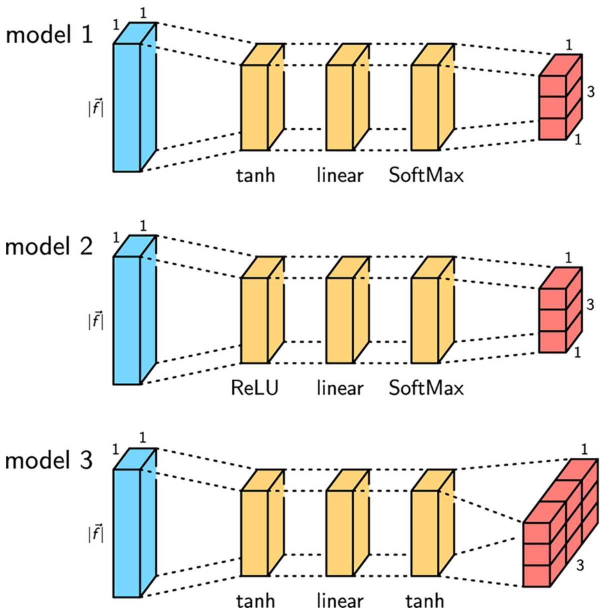

Fig. 7. Architectures of the 3 deep learning model used to predict the most optimal net ordering.

In total, we propose 3 deep learning models for the task of identifying the optimal net ordering for a single layer routing solution S^1. These 3 deep learning models with their layer configurations and loss functions are summarized in Table 1 and are illustrated in Figure 7. We selected a basic deep learning architecture, a multilayer perceptron (MLP), to ensure accessibility and simplicity. The MLP is a fundamental neural network model that is widely understood and can be easily adapted for various use cases. In order to predict the net ordering value through deep learning, we utilized two types of output layers: a softmax activation function for Models 1 and 2, and a hyperbolic tangent (tanh) function for Model 3. Model 1 and Model 2 return values for $\text{optimality}(S_h^k)$ for each net ordering no_h in a data group corresponding to a solution S^1. Based on the value for $\text{optimality}(S_h^k)$, the optimal net ordering is determined within a data group associated to S^1. Here, the MSE defined in (12) is used to minimize the difference between the actual best net ordering with the predicted net ordering during the model training process. In comparison, Model 3 returns a probability value for each net ordering no_h in a data group corresponding to S^1. The probability is a measure of whether the predicted no_h is the actual optimal net ordering for S^1. Using the cross-entropy defined in (13), the model is trained to maximize the probability q_{ij} of the predicted optimal net ordering no_j across different data groups corresponding to different single layer solutions S_i^1. Furthermore, in order to explore the impact of different activation functions in the first layer, Model 1 was chosen to use the ReLU function and Model 2 was chosen to use the tanh function for comparison.

Experiment and data

Data preparation

Figure 8 briefly shows the process of generating datasets and how to use the generated datasets as input data in our deep learning model. In the dataset preparation part, we computationally generate the 16 independent training datasets before training the deep learning model. One dataset includes $N = 2500$ data groups, where each data group corresponds to a different 1-layer routing solution S_i^1 for a corresponding randomly generated routing problem. For a given data group, we first randomly choose a constant number of vertices per layer and assign the selected vertices to the number of nets N_{nets} at random. Secondly, we find the 1-layer routing solution S^1 through layer compression using 1-layer routing algorithms such as KA, and ST. Thirdly, the feature vector

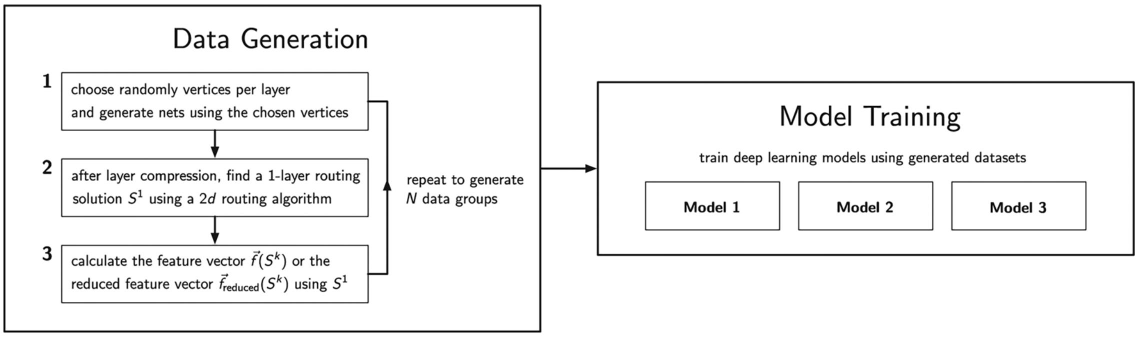

Fig. 8. The data preparation process and the training process for the deep learning model.

Dataset parameter	Options
1-layer router type	Kruskal algorithm(KA), Steiner tree(ST)
number of layers k	2, 5
number of nets N_{nets}	3, 5
feature vector	all features ($\vec{f}$), reduced features($\vec{f}_{\text{reduced}}$)

Table 2. Summary of parameters that define the routing problem conditions for each dataset during dataset generation.

$\vec{f}(S^k)$ or reduced feature vector $\vec{f}_{\text{reduced}}(S^k)$ of the 1-layer 2d routing solution S^1 is calculated. Finally, after repeating the above steps for N times, a single dataset is generated. We then move on to training our 3 deep learning summarized in Table 2 and illustrated in Figure 7 using the feature vector $\vec{f}(S^k)$ or reduced feature vector $\vec{f}_{\text{reduced}}(S^k)$ generated previously as part of the dataset.

For training, we use in total 16 independent generated datasets that are determined by 4 conditions, which are varied as discussed as follows:

- Number of layers k : We vary the total number of layers k in the routing environment. The 16 datasets are generated such that for 8 datasets we have $k = 2$ layers in the routing environment and for the other 8 datasets we have $k = 5$ layers in the routing environment.
- Number of nets N_{nets} : We vary the total number of nets N_{nets} in the k-layer routing problem in order to generate the 16 independent datasets. 8 of the datasets are obtained by setting $N_{\text{nets}} = 3$, while the other remaining 8 datasets are obtained by setting $N_{\text{nets}} = 5$.
- Single Layer Router: Before layer assignment, we have to first have a 1-layer routing solution S^1 with a choice of net ordering *no*. The 1-layer routing solution S^1 is found by one of two algorithms, the first being the Kruskal's algorithm (KA) and the second being the Steiner Tree algorithm (ST). 8 of the datasets have been generated using Kruskal's algorithm and the other 8 datasets have been generated using the Steiner Tree algorithm for S^1.
- Selection of routing features:we can assign a feature vector $\vec{f}(S_h^k)$ for a given k-layer routing solution S_h^k obtained from a net ordering no_h as defined in (9). This feature vector contains as components the number of pins $|n_m^k|$ in the k-layer environment, the number of pins $|n_m^1|$ in the single layer environment, the number of vertices $|V_m^1|$ in the single layer environment, the overflow $o(E_m^1)$ in the single layer environment, the number of branch vertices $|V_{\text{branch}}^1|$ in the single layer environment, and information on the minimal encompassing rectangle as defined in (8) for the single layer environment. We note that most of the features in $\vec{f}(S_h^k)$ depend on the single layer routing solution S^1 rather than information relating to the k-layer routing problem.

As argued in[30–32], a proper optimal selection of features often leads to a better training result for deep learning models. Accordingly, when we generate the training datasets, we also allow an option for a narrower selection of features. Being always motivated to choose features from the single layer routing solution S^1 that highly correlate with features of the k-layer routing solution S_h^k for a given net ordering no_h, we select a subset of features[33,34] that correspond to the 1-layer solution S^1. This subset forms a reduced feature vector $\vec{f}_{\text{reduced}}(S_h^k)$ for a given k-layer routing solution S_h^k, which we define as follows

$$\vec{f}_{\text{reduced}}(S_h^k) = \left(\{|n_m^k|\}, \{|n_m^1|\}, \{|V_m^1|\}, \{o(E_m^1)\}\right) \ , \tag{14}$$

where $m = 1, \ldots, N_{\text{pins}}$ is the index over nets n_m^1 in the single layer environment. For 8 datasets, we calculate $\vec{f}(S_h^k)$ as the feature vector, and for the remaining 8 dataset we use the reduced feature vector $\vec{f}_{\text{reduced}}(S_h^k)$.

Table 2 summarizes the above parameters.

A complete summary of the 16 datasets generated by varying the parameters described above is given in Table 3. Given a dataset generated for a combination of fixed conditions, as described above, we generate $N = 2500$ data groups each corresponding to a 1-layer routing solution for randomly generated routing problems. The routing problems are restricted to a *k*-layer routing environment where each layer consists of a 5×5 grid graph G^1. Furthermore, the vertices v_m^k that correspond to pins in the routing problem and are part of the connected graph in the routing solution are restricted such that the number of them is a constant 15 per layer. This ensures that in the generated datasets are made of *k*-layer routing solutions S^k that evenly involve 15 connected vertices v_m^k per layer. The 1-layer routing solution S^1 of one sample data group in Data 3 is shown in Figure 9.

Experiments and model training

We divide the generated dataset into a 80% training set and a 20% testing set. The test data is used to evaluate the accuracy of our proposed deep learning models. Our deep learning models were trained by using the Adam optimizer and by varying various hyperparameter values of the models. For hyperparameter tuning, a grid search was used to train our models while changing each hyperparameter value by a certain amount. This approach allows us to identify optimal configurations that enhance model performance and generalizability. By exploring the hyperparameter space, we aimed to mitigate the risk of overfitting, ensuring that our models not only perform well on the training data but also exhibit robust performance on unseen data. Table 4 summarized the values of each hyperparameter that was used for hyperparameter tuning during model training.

Training accuracy is calculated based on the performance of our deep learning models within each data group in the generated database. For a given dataset we have $N = 2500$ data groups that form a set of different routing problems in the same routing environment. As a result, each data group is associated with a unique 1-layer routing solution S^1. The actual optimal net ordering *no* of a given S^1 corresponding to a data group is obtained using the simplified criteria summarized above. The accuracy of our deep learning models depends on whether the actual optimal net ordering *no* for S^1 corresponding to a given data group is matched by the predicted optimal net ordering $\hat{no}$ for S^1 obtained from one of our deep learning models. Accordingly, the overall accuracy of the deep learning model can be calculated following,

$$\text{accuracy}(\%) = \frac{n_{\text{match}}}{N} \quad (\times 100\%) \ , \tag{15}$$

where N is the number of data groups defined on the same routing problem parameters, and n_{match} is the number of data groups for which the machine learning predicted the correct optimal net ordering $\hat{no} = no$. Table 5 illustrates an example calculation of the model accuracy for $N = 5$ data groups. Figure 10 shows a selection of loss curves during the training of models 1 and 2 with selected training hyperparameters.

Results

First, our experiments on our trained deep learning models suggest that our deep learning models perform better in predicting the optimal net ordering based on the optimality criteria than the heuristic score function

Dataset	1-layer router type	*k*-layers	N_{nets}	Features
Data 1	KA	2	3	all
Data 2	KA	2	3	reduced
Data 3	ST	2	3	all
Data 4	ST	2	3	reduced
Data 5	KA	5	3	all
Data 6	KA	5	3	reduced
Data 7	ST	5	3	all
Data 8	ST	5	3	reduced
Data 9	KA	2	5	all
Data 10	KA	2	5	reduced
Data 11	ST	2	5	all
Data 12	ST	2	5	reduced
Data 13	KA	5	5	all
Data 14	KA	5	5	reduced
Data 15	ST	5	5	all
Data 16	ST	5	5	reduced

Table 3. Summary of generated dataset with choice of parameters that are used to generate the sample routing solutions and their features.

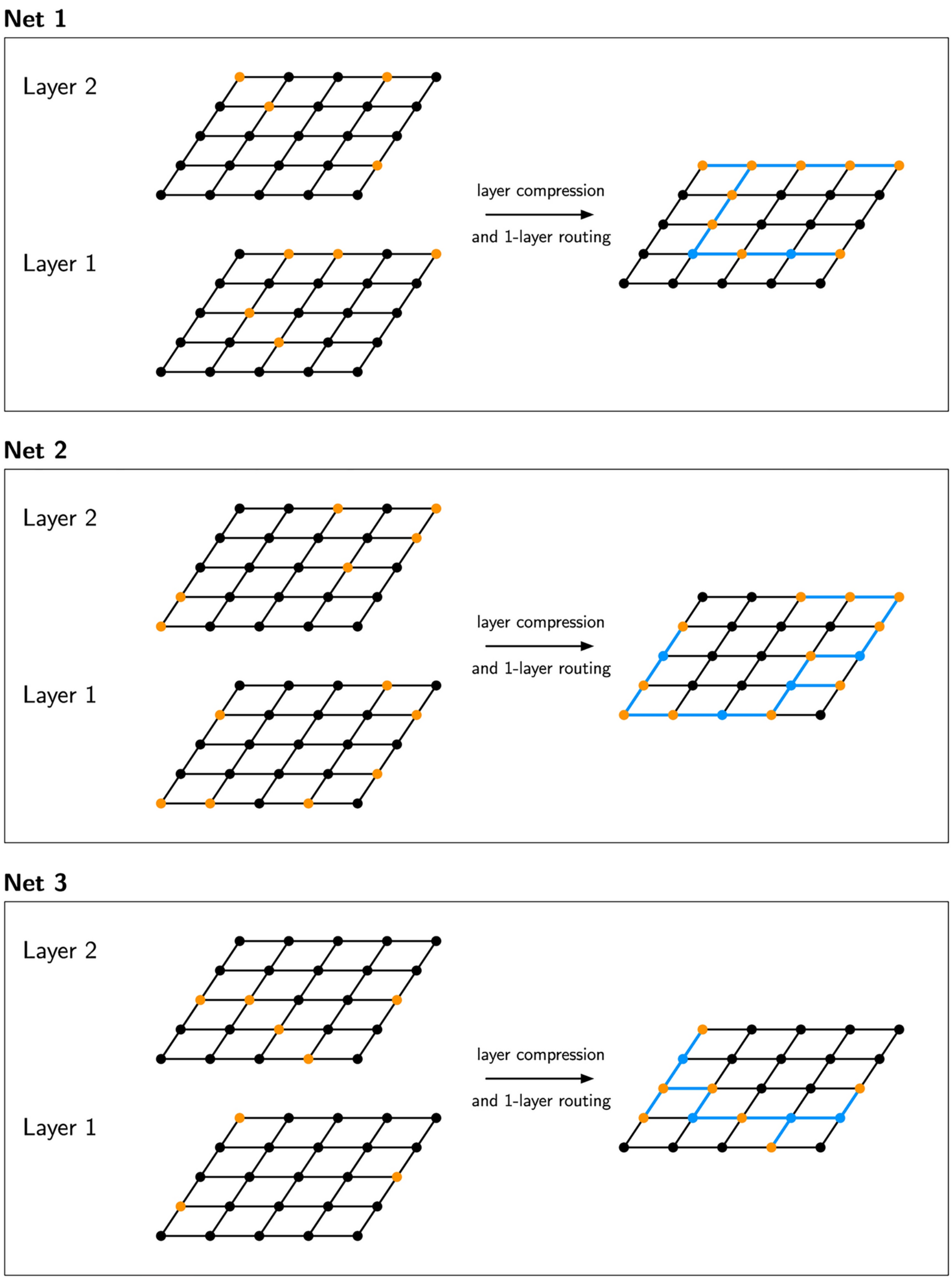

Fig. 9. 1-layer routing solution S^1 for a $k=2$ layer routing environment with $N_{\text{nets}}=3$. The vertices v_m^k corresponding to pins are highlighted in orange and are restricted to a constant number of 15 per layer. The 1-layer routing solution S^1 was obtained through the Steiner tree algorithm.

Hyperparameter	Values
Number of epochs	30, 50, 70, 90, 100, 150, 200, 500, 1000, 1500, 2000
Number of units	10, 20, 30 ,40, 50, 60, 70, 80, 100
Learning rate	0.0001, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.008, 0.01

Table 4. Hyperparameter values used during hyperparameter tuning of the 3 deep learning models. The unit number corresponds to the layer width of the hidden layers in the neural network.

Data group	Predicted order $\hat{no}$	Actual order *no*	Match	Accuracy
1	(1, 2, 3)	(1, 2, 3)	○	60%
2	(3, 1, 2)	(3, 1, 2)	○	
3	(1, 3, 2)	(1, 2, 3)	×	
4	(2, 1, 3)	(3, 2, 1)	×	
5	(1, 3, 2)	(1, 3, 2)	○	

Table 5. An example for calculating the accuracy for $N = 5$ data groups corresponding to the same dataset.

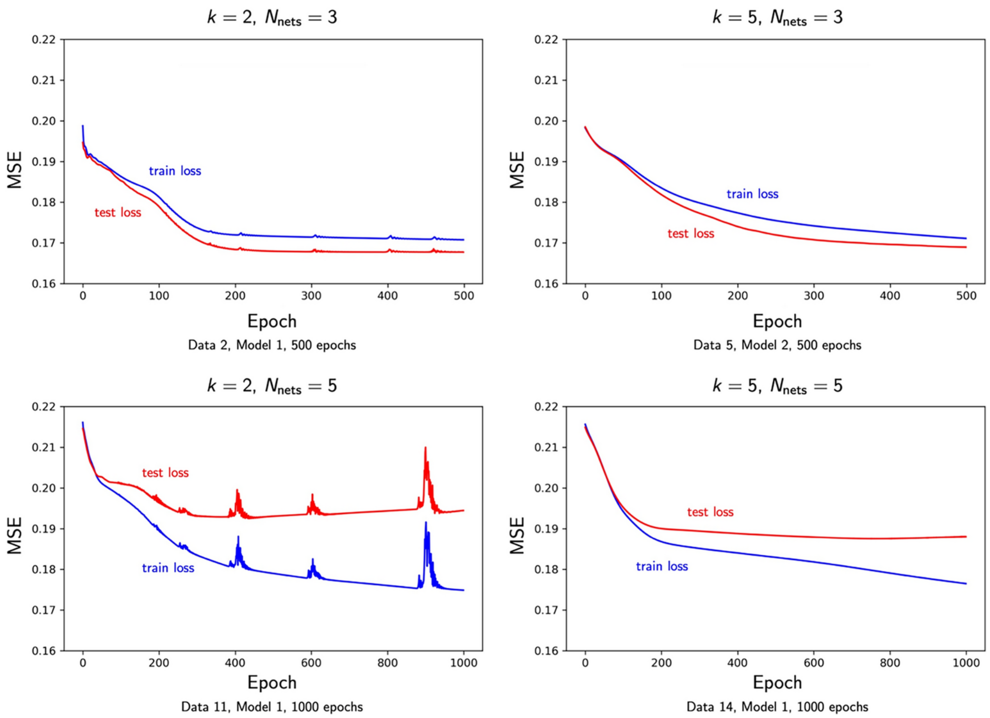

Fig. 10. A selection of loss curves during training for a some proposed deep models with training hyperparameters.

in[2] or even a random choice of net ordering. Figure 11 and Figure 12 show a selection of a 2-layer and a 5-layer routing result respectively using our proposed deep learning models for net ordering. Table 6summarizes the average accuracies of the predicted optimal net orderings for the 3 proposed deep learning models, the heuristic method based on[2] and a random generator of net orderings. Note that the random generator independent of the 1-layer router type and layer number *k* generates a random net order *no* which is only dependent on the number of nets N_{nets}. Furthermore, the heuristic method only uses the reduced set of features and therefore the average accuracies for the heuristic method do not distinguish between datasets with reduced or full set of features. The average accuracies are calculated over the $N = 2500$ data groups for each of the 16 datasets. Table 6 shows the highest average accuracies for a given *k*-layer environment and N_{nets} number of nets in bold.

For all datasets, our proposed deep learning models show much higher accuracy than the heuristic score function from[2] or the random generator. When the number of nets is $N_{\text{nets}} = 3$, the performance of our deep learning models is more than twice as the heuristic method or the random generator. Furthermore, when the number of nets is $N_{\text{nets}} = 5$, the performance of our deep learning models is more than 10 times higher than the heuristic method, which with higher net numbers performs nearly as bad as a simple random generator. Overall, as expected, smaller numbers of nets and layers decreases the complexity of the routing problem, resulting in an overall increase in the accuracy of the predicted optimal net ordering by our deep learning models.

Between the 3 different deep learning models that we propose in this work, models 1 and 2 perform better on average across all 16 datasets than model 3. This may indicate that the use of MSE as the loss function is better than the use of cross-entropy as suggested in Section §3. Moreover, according to the results in Table 6, it appears that the 3 deep learning models on average performed better across all the 16 datasets when the complete feature vector $\vec{f}(S^k)$ was used rather than the reduced feature vector $\vec{f}_{\text{reduced}}(S^k)$. This indicates that the more we know about the 1-layer routing result S^1, the better our deep learning models perform in identifying the most optimal net ordering no_h that results in the most optimal *k*-layer routing solution S_h^k. We also note that overall

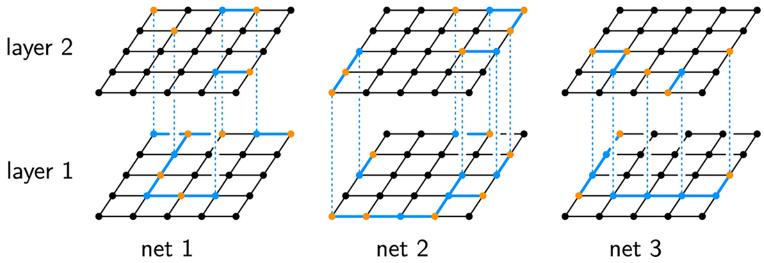

Fig. 11. Routing result S^k for a $k = 2$ layer routing environment with $N_{\text{nets}} = 3$. The vertices v_m^k that corresponding to pins are highlighted in orange and are restricted to a constant number of 15 per layer. The 1-layer routing solution S^1 was obtained through the Steiner tree algorithm.

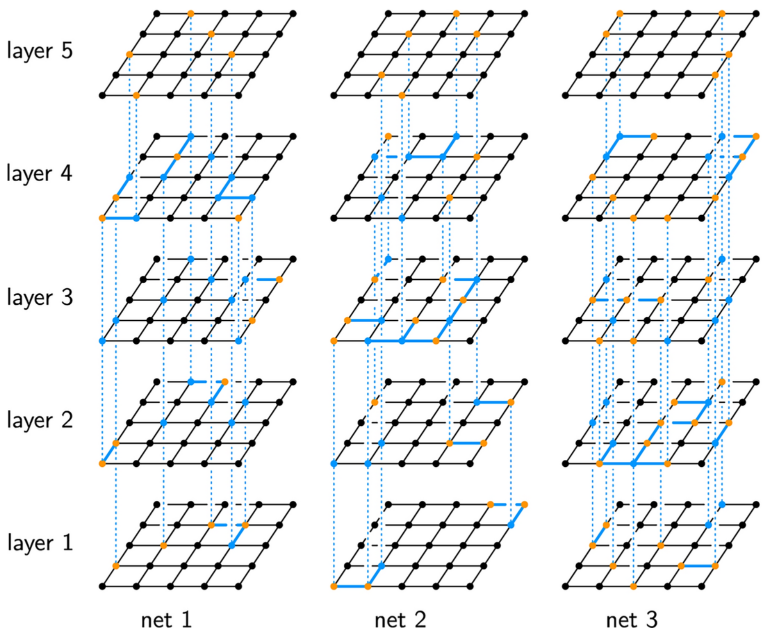

Fig. 12. Routing result S^k for a $k = 5$ layer routing environment with $N_{\text{nets}} = 3$. The vertices v_m^k that corresponding to pins are highlighted in orange and are restricted to a constant number of 15 per layer. The 1-layer routing solution S^1 was obtained through the Steiner tree algorithm.

the results based on the Kruskal algorithm (KA) for the 1-layer router perform better than the results based on the Steiner tree (ST) algorithm.

Because our deep learning models significantly performed better than the previous heuristic score function method, we tested whether our trained deep learning models are transferable to other routing environments and problems. In order to test transferability[35,36], we select the best performing deep learning models in Table 6 according to datasets that have 3 nets. Deep learning models 1 and 2 are trained using respectively datasets Data 2 and Data 1 in Table 3 with 2 layers and 3 nets, and model 2 is trained with dataset Data 5 with 5 layers and 3 nets as defined in Table 3. These deep learning models performed based for these specific datasets according to the results in Table 6. We then used these 3 trained models and calculated their average accuracy on datasets with the same number of nets, but layer numbers ranging from 2 to 10. Figure 13 summarizes the resulting average accuracy values for the 3 deep learning models, and we can see that there is no significant performance drop with increasing number of layers in the test datasets. This is even the case for model 2 trained on dataset Data 5, which with 5 layers and 3 nets is highly complex. As a result, our proposed deep learning models appear to be highly transferable to routing environments with the same number of nets, but increasingly larger number of layers. This underlines the significant performance increase by our deep learning models and we hope to report on further improvements based on our deep learning models in the near future.

Conclusions

In this work, we have proposed a new net ordering method based on machine learning techniques. Our new method outperforms previous heuristics-based methods proposed in[2]based on the experiments conducted as part of this work. Moreover, our experiments have also shown that indeed net ordering significantly affects the

	Accuracy(%)				
Dataset	**Model 1**	**Model 2**	**Model 3**	**Heuristic**	**Random**
Data 1	40.69	**41.30**	36.63		16.67
Data 2	**41.30**	40.89	33.82	17.25	
Data 3	39.72	39.11	36.90		
Data 4	40.93	40.93	38.10	15.44	
Data 5	40.44	**42.05**	34.21		
Data 6	40.04	41.25	33.20	14.41	
Data 7	36.87	37.27	32.46		
Data 8	36.47	38.08	31.46	14.51	
Data 9	7.22	**9.69**	4.12		0.83
Data 10	7.63	7.84	3.30	0.70	
Data 11	8.81	7.79	2.66		
Data 12	8.40	8.20	2.87	0.82	
Data 13	7.58	7.99	4.11		
Data 14	7.38	7.58	2.87	0.25	
Data 15	7.76	**8.16**	4.29		
Data 16	7.35	7.55	4.08	0.78	

Table 6. Summary of accuracy measurements for the proposed 3 deep learning models, the heuristic score function method based on previous work and a simple random generator. The accuracy measurements were performed over 16 datasets that each contain $N = 2500$ randomly generated data groups associated to routing problems set under the same routing and environmental parameters specific to the dataset.

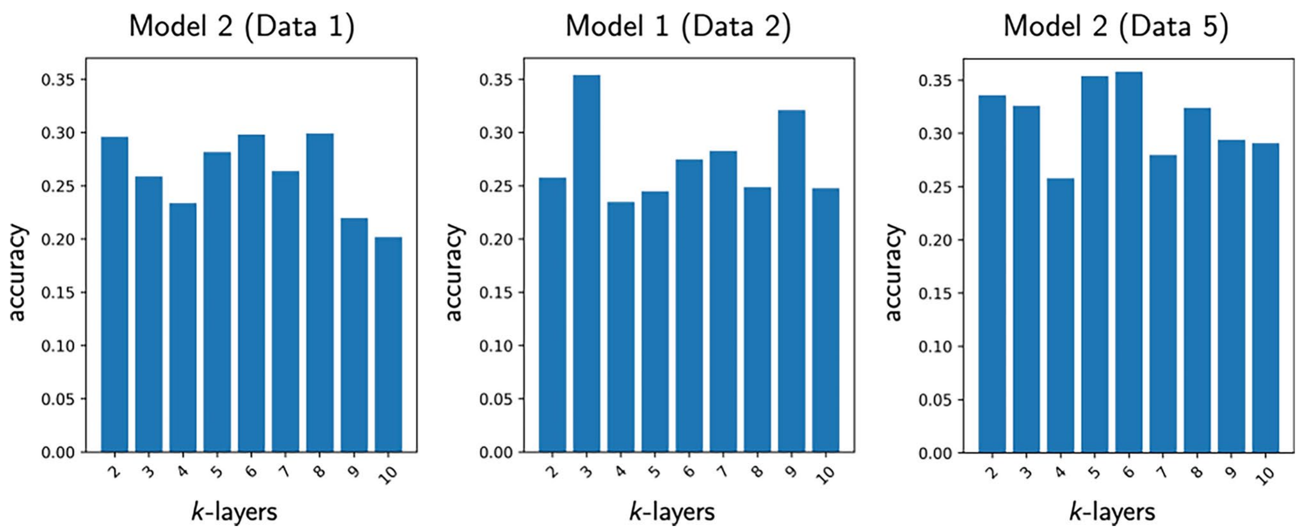

Fig. 13. Models trained on a single dataset perform well for routing environments with different number of layers.

overall layer assignment process of the 2-dimensional routing result in global routing. As a result, our machine learning based method significantly improved the global routing algorithm first proposed in[2] for semiconductor chip and package design.

As discussed in section §2, net ordering in global routing with layer compression plays a crucial role in the global routing process in semiconductor package design. We however emphasize that our work does not optimize other optimizeable steps in global routing with layer compression, including the algorithm for $2d$ routing after layer compression or the discretization of the routing environment. Our work concentrates on optimizing the net ordering part of the global routing process with layer compression. A small improvement as we achieved in this work in the sequence of optimizeable steps in the global routing process has a cumulative positive effect for the overall global routing result.

To give an example, we expect to be able to further optimize the global routing process by applying machine learning techniques to the actual layer assignment process described in[2]. like net ordering, the algorithm in[2] for layer assignment of the 2-dimensional routing results is based on a heuristics-based minimum via cost function that measures the cost of lifting a certain segment of the 2-dimensional routing result to a k-th layer by assessing the cost of adding a via connection from layer 1 to layer k. This heuristics-based minimum cost function indeed can be replaced by a machine learning model that optimizes the minimum cost much more effectively than a hard-coded cost function based on heuristics. As a result, we believe that our improvements achieved in this work can be combined with further optimized routing steps such as the 2d routing step or the routing environment discretization step. We hope to explore these improvements in future work.

Additionally, we expect that further optimization can be achieved in net ordering if the number of layers and nets increases beyond the scope of the training data that we have used as part of this work. Furthermore, we hope to further improve our machine learning model by testing it against other model architectures such as convolutional neural networks (CNN) and graph neural networks (GNN). We hope to explore these directions in future work.

As a final comment, we would like to emphasize that semiconductor package design heavily relies on physical design constraints. These may include the efficiency of the design in dispersing heat generated by the components of the semiconductor package and the minimum distance between connections in order to avoid signal interference. Evaluating the overall performance of a particular routing design under the collection of these physical constraints in itself is a challenging problem, which we plan to explore in future work.

Data availibility

The data generated or analyzed during this study are available from the corresponding author on reasonable request.

Received: 11 May 2024; Accepted: 3 December 2024
Published online: 28 December 2024

References

1. Gao, J.-R., Wu, P.-C. & Wang, T.-C. A new global router for modern designs. In *2008 Asia and South Pacific Design Automation Conference*, 232–237 (IEEE, 2008).
2. Lee, T.-H. & Wang, T.-C. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27**, 1643–1656 (2008).
3. Chang, Y.-J., Lee, Y.-T., Gao, J.-R., Wu, P.-C. & Wang, T.-C. Nthu-route 2.0: a robust global router for modern designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **29**, 1931–1944 (2010).
4. Lee, T.-H. & Wang, T.-C. Robust layer assignment for via optimization in multi-layer global routing. In *Proceedings of the 2009 international symposium on Physical design*, 159–166 (2009).
5. Kahng, A. B. Machine learning for cad/eda: the road ahead. *IEEE Design & Test*. **40**, 8–16 (2022).
6. Chan, W.-T. J., Ho, P.-H., Kahng, A. B. & Saxena, P. Routability optimization for industrial designs at sub-14nm process nodes using machine learning. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, 15–21 (2017).
7. Lim, S. K. *Practical problems in VLSI physical design automation* (Springer Science & Business Media, 2008).
8. Wang, L.-T., Chang, Y.-W. & Cheng, K.-T. T. *Electronic design automation: synthesis, verification, and test* (Morgan Kaufmann, 2009).
9. Lavagno, L., Markov, I. L., Martin, G. & Scheffer, L. K. *Electronic design automation for IC implementation, circuit design, and process technology: circuit design, and process technology* (CRC Press, 2016).
10. Lavagno, L., Scheffer, L. & Martin, G. *EDA for IC implementation, circuit design, and process technology* (CRC press, 2018).
11. Huang, G. et al. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*. **26**, https://doi.org/10.1145/3451179 (2021).
12. Albrecht, C. Global routing by new approximation algorithms for multicommodity flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **20**, 622–632 (2001).
13. Wu, T.-H., Davoodi, A. & Linderoth, J. T. Grip: Global routing via integer programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **30**, 72–84 (2010).
14. Behjat, L., Vannelli, A. & Rosehart, W. Integer linear programming models for global routing. *INFORMS Journal on Computing*. **18**, 137–150 (2006).
15. Chen, H.-Y., Hsu, C.-H. & Chang, Y.-W. High-performance global routing with fast overflow reduction. In *2009 Asia and South Pacific Design Automation Conference*, 582–587 (IEEE, 2009).
16. Cho, M., Lu, K., Yuan, K. & Pan, D. Z. Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*. **14**, 1–21 (2009).
17. Moffitt, M. D. Maizerouter: Engineering an effective global router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **27**, 2017–2026 (2008).
18. Yutsis, V., Bustany, I. S., Chinnery, D., Shinnerl, J. R. & Liu, W.-H. Ispd 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement. In *Proceedings of the 2014 on International symposium on physical design*, 161–168 (2014).
19. Zhou, Q. et al. An accurate detailed routing routability prediction model in placement. In *2015 6th Asia Symposium on Quality Electronic Design (ASQED)*. 119–122 (IEEE, 2015).
20. Qi, Z., Cai, Y. & Zhou, Q. Accurate prediction of detailed routing congestion using supervised data learning. In *2014 IEEE 32nd international conference on computer design (ICCD)*. 97–103 (IEEE, 2014).
21. Spindler, P. & Johannes, F. M. Fast and accurate routing demand estimation for efficient routability-driven placement. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, 1–6 (IEEE, 2007).
22. Hsu, C.-H., Chen, H.-Y. & Chang, Y.-W. Multi-layer global routing considering via and wire capacities. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, 350–355 (IEEE, 2008).
23. Wu, D., Hu, J., Mahapatra, R. & Zhao, M. Layer assignment for crosstalk risk minimization. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*, 159–162 (IEEE, 2004).
24. Ciesielski, M. J. Layer assignment for vlsi interconnect delay minimization. *IEEE transactions on computer-aided design of integrated circuits and systems*. **8**, 702–707 (1989).
25. Sechen, C. & Sangiovanni-Vincentelli, A. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*. **20**, 510–522 (1985).
26. Shin, H. & Sangiovanni-Vincentelli, A. Mighty: a rip-up and reroute detailed router. *In Proc. ICCAD*. **86**, 2–5 (1986).
27. Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*. **7**, 48–50 (1956).
28. Kou, L., Markowsky, G. & Berman, L. A fast algorithm for steiner trees. *Acta informatica*. **15**, 141–145 (1981).
29. Nam, G.-J., Sze, C. & Yildiz, M. The ispd global routing benchmark suite. In *Proceedings of the 2008 international symposium on Physical design*, 156–159 (2008).

30. Jurado, S., Nebot, À., Mugica, F. & Avellana, N. Hybrid methodologies for electricity load forecasting: Entropy-based feature selection with machine learning and soft computing techniques. *Energy.* **86**, 276–291 (2015).
31. Zheng, J., Yang, W. & Li, X. Training data reduction in deep neural networks with partial mutual information based feature selection and correlation matching based active learning. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2362–2366 (IEEE, 2017).
32. Hwang, J. K., Yun, G. Y., Lee, S., Seo, H. & Santamouris, M. Using deep learning approaches with variable selection process to predict the energy performance of a heating and cooling system. *Renewable Energy*. **149**, 1227–1245 (2020).
33. Jebli, I., Belouadha, F.-Z., Kabbaj, M. I. & Tilioua, A. Prediction of solar energy guided by pearson correlation using machine learning. *Energy*. **224**, 120109 (2021).
34. Luong, H. H. et al. Feature selection using correlation matrix on metagenomic data with pearson enhancing inflammatory bowel disease prediction. In *International Conference on Artificial Intelligence for Smart Community: AISC 2020, 17–18 December, Universiti Teknologi Petronas, Malaysia*, 1073–1084 (Springer, 2022).
35. Torrey, L. & Shavlik, J. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, 242–264 (IGI global, 2010).
36. Zhuang, F. et al. A comprehensive survey on transfer learning. *Proceedings of the IEEE*. **109**, 43–76 (2020).

Acknowledgements
The authors were supported by a UNIST Industry Research Project (2.220916.01) funded by Samsung SDS in Korea, as well as a UNIST UBSI Research Fund (1.220123.01, 1.230065.01) and the BK21 Program ("Next Generation Education Program for Mathematical Sciences", 4299990414089) funded by the Ministry of Education in Korea and the National Research Foundation of Korea (NRF). C.-H. L. is supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2022R1F1A1064487). R.-K. S. is supported by a Basic Research Grant of the National Research Foundation of Korea (NRF-2022R1F1A1073128). He is also supported by a Start-up Research Grant for new faculty at UNIST (1.210139.01) and a UNIST AI Incubator Grant (1.230038.01).

Author contributions
H. C. and M. L. have contributed equally to this work. All authors reviewed the manuscript.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to R.-K.S.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.