The DeepFogSim Simulator

A quick guide

Michele Scarpiniti [⋆] , **Enzo Baccarelli** , **Alireza Momenzadeh**

December 2020, Rome
Version 4.0

[⋆] **Correspondence: Michele Scarpiniti**
Department of Information Engineering, Electronics and Telecommunications (DIET), Sapienza University of Rome
Web page: http://michelescarpiniti.site.uniroma1.it
Email: michele.scarpiniti@uniroma1.it

FOREWORD: The Deep Fog Simulator (*DeepFogSim*) is a software toolbox aiming at testing the performance of virtualized technological platforms for the real-time distributed execution of the inference phase of CDNNs with early exits under IoT realms. The main peculiar features of the proposed *DeepFogSim* toolbox are that: (i) it allows the *joint dynamic* energy-aware optimization of the Fog-hosted computing-networking resources under *hard constraints* on the tolerated inference delays; (ii) it allows the *repeatable* and *customizable* simulation of the resulting energy-delay performance of the overall Fog execution platform; (iii) it allows the *dynamic tracking* of the *performed resource allocation* under time-varying operating conditions and/or failure events; (iv) it is equipped with a user-friendly *Graphic User Interface* (GUI) that supports a number of graphic formats for data rendering.
The software platform utilized for the *DeepFogSim* development is MATLAB. The overall simulator script is composed by about 5,000 lines of MATLAB code. The MATLAB release 2015 (or later) is recommended for running the *DeepFogSim* code.
The *DeepFogSim* toolkit has been developed by Profs. Michele Scarpiniti and Enzo Baccarelli at the "Department of Information Engineering, Electronics and Telecommunications" (DIET) of "Sapienza" University of Rome (Italy).

1 DEEPFOGSIM – THE CONSIDERED FOG NETWORKED INFRASTRUCTURE

The recent convergence of the Mobile-Fog-Cloud technological platforms allows the real-time and energy-efficient execution of analytics required by future Internet applications [1]. Among these, the recent introduction of the so-called Conditional Neural Networks (CDNNs) with multiple early exits, executed atop virtualized multi-tier Fog platforms, makes feasible fast prediction on input data and a more efficient use of the whole technological platform [2]. In this scenario, it is really important to evaluate, test and compare the performance of the computing-networking building blocks supporting the CDNN execution.

In order to guarantee such a distributed implementation, the technological platform supporting the execution of a CDNN with early exits cannot rely only on a standing-alone remote and centralized cloud data center. It should be composed, indeed, of the networked interconnection of a number of hierarchically-organized computing nodes, which are spatially scattered and operate nearby the IoT devices. This is the native layout of the emerging paradigm of Fog Computing (FC) [1]. It enables pervasive local access to virtualized small-size pools of computing resources that can be quickly provisioned, dynamically scaled up/down and released on an on-demand basis. Nearby resource-limited mobile devices may access these resources by establishing single-hop WiFi-supported communication links. Like the Cloud paradigm, the Fog one exploits resource virtualization for supporting a set of virtualized services. In this regard, FC complements Cloud Computing (CC) by distributing computing-plus-communication virtualized resources along the full path from the IoT realm to the remote Cloud data center [1] and offers a powerful paradigm for developing DL applications [3].

According to these considerations, a FC technological platform is typically composed of a set of $(M-1)$ clusters of medium-size virtualized data centers (i.e., the Fog Nodes (FNs)), which are hierarchically-organized into tiers and exploit (typically wireless) transport links for enabling inter-tier communication. A sketch of a multi-tier networked Fog platform is shown in Fig. 1b.

In order to describe more in depth the FC platform and the application that should run atop it, we assume that a data vector $\vec{z}_0(t)$, arriving at the input of the CDNN of Fig. 1a at time t, is processed in a layer-wise way, moving from $layer\#1$, in order to generate a local class label (i.e., a local decision). If the confidence level computed by the local classifier at $layer\#l$, $1 \leq l \leq L-1$, is high enough, the processing is stopped and a decision is output from the l-th branch of Fig.1a. In the opposite case, the feature vector $\vec{z}_l(t)$ extracted by $layer\#l$ is passed to the next $layer\#(l+1)$ for further processing. Only when an input data is so challenging to classify to require the processing by the full CDNN stack, the corresponding decision is generated by the last L-th layer, irrespective of its actual confidence level.

1.1 THE RESULTING IOT-FOG-CLOUD EXECUTION PLATFORM

As stated, the FC platform is suitable to execute CDNNs with early exits [2, 4]. The resulting FC technological platform is typically composed of a set of $(M-1)$ clusters of medium-size virtualized data centers (i.e., Fog Nodes (FNs)), which are hierarchically-organized into tiers and exploit (typically wireless) transport links for enabling inter-tier communication. Doing so, a communication path is implemented from the lowermost IoT realm at $tier\#0$ to the uppermost Cloud Node (CN) at $tier\#M$. At each intermediate $tier\#m$, with $1 \leq m \leq M-1$, an intra-tier local network allows an AGgregator (AG) node to provide an early exit by suitably merging the outputs of the corresponding FNs into a local output (see the per-tier side branches in Fig. 1b).

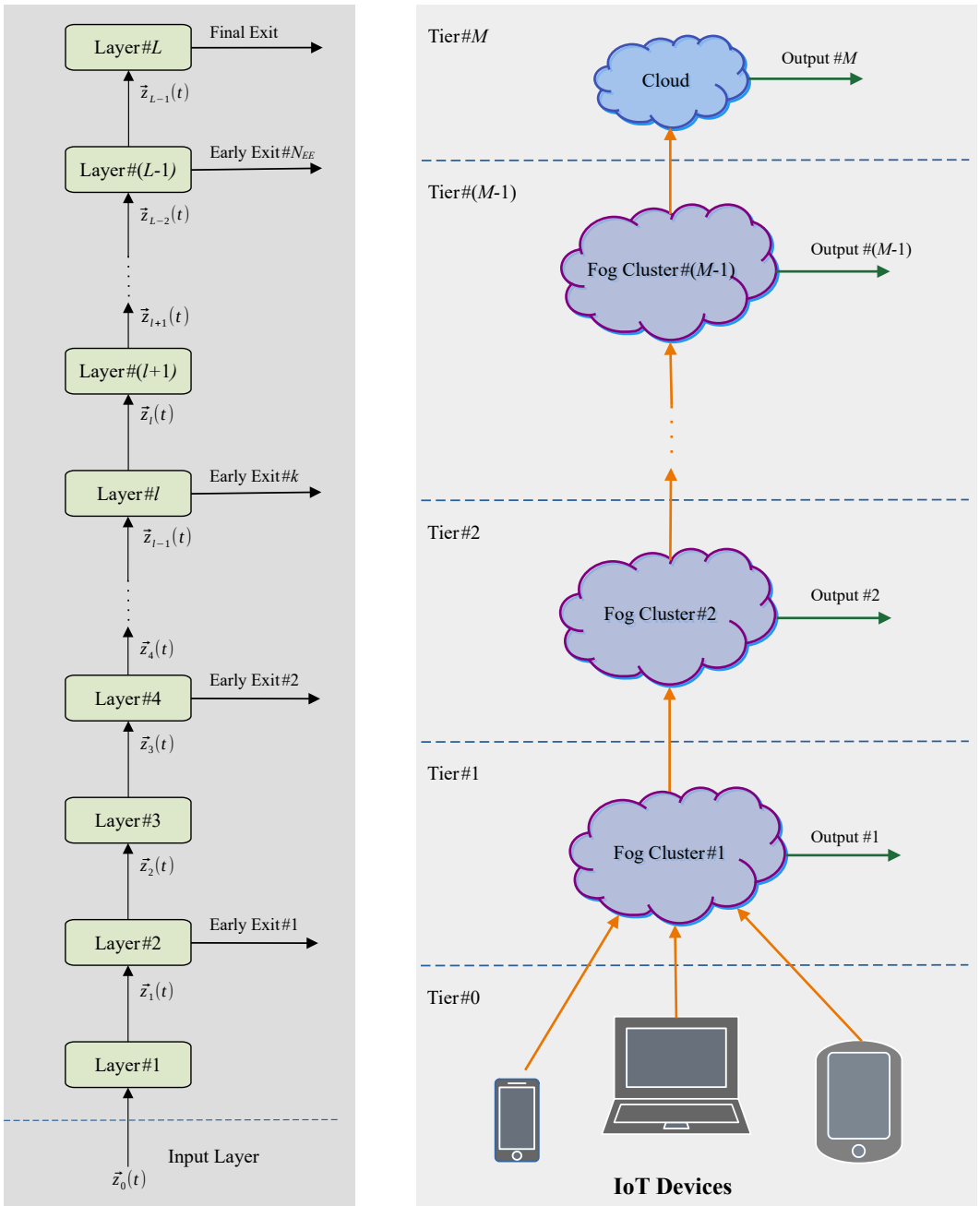


FIGURE 1. (a) Sketch of the stack topology of a CDNN with L layers and N_{EE} early exits; (b) Sketch of a multi-tier networked Fog platform.

Doing so, it is expected that only a small fraction of the volume $v_0(t)$ of data generated by the IoT devices at time t needs to be transported up to the remote CN for analytics, while a (hopefully) large part of $v_0(t)$ early exits through the available intermediate per-tier local outputs. Fig. 2 shows a sketch of the envisioned multi-tier networked technological platform for the distributed execution of the inference phase of an (already designed and trained) CDNN with early exits.

Basically, the Fog platform of Fig. 2 is composed of the hierarchically organized cascade of three main segments, namely:

1. the *lowermost* segment (i.e., *tier* #0), where a set of spatially distributed resource-poor IoT sensors operate. Since current IoT devices (for example, smartphone, tablets and personal assistants, just to name a few) are natively equipped with built-in sensors, IoT devices at *tier* #0 are assumed to be co-located with Fog nodes at *tier* #1 (see the dark blue triangles at the bottom of Fig. 2);

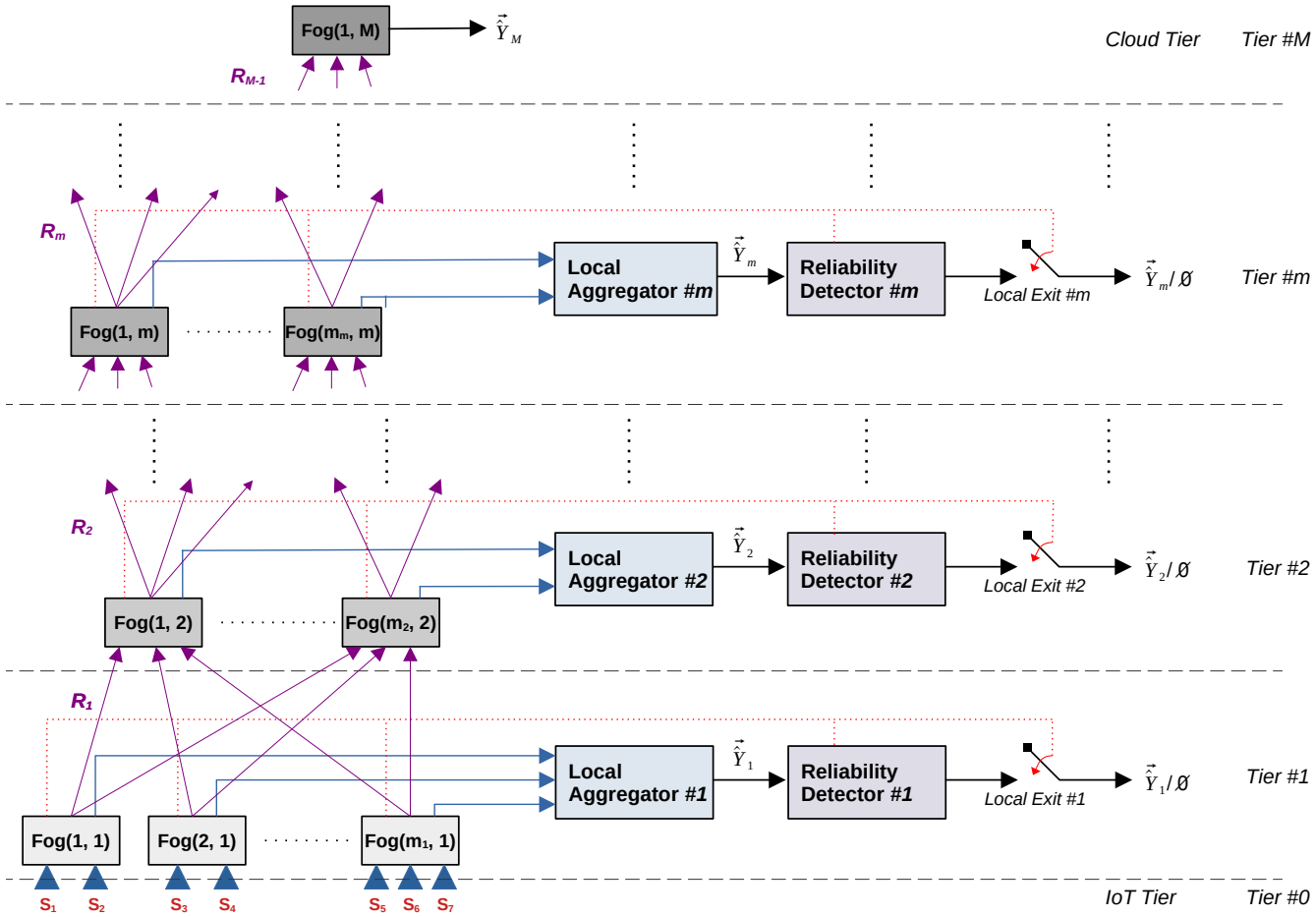


FIGURE 2. The IoT-Fog-Cloud execution platform simulated by *DeepFogSim*.

2. the *middle* segment, where a number of spatially distributed and networked Fog nodes operates. According to Fig. 2, this segment is hierarchically organized into $(M-1)$ stacked tiers numbered from $tier\#1$ to $tier\#(M-1)$. At $tier\#m$, with $1 \leq m \leq M-1$, a cluster: $\{\text{Fog}(j,m),\ 1 \leq j \leq m_m\}$, with $m_m \geq 1$, of Fog nodes operates by exchanging data with a local aggregator $AG(m)$ over an intra-tier LAN. From time to time, the local $Detector\#m$ generates a local decision $\hat{c}_m$ (i.e., a class label) on the current input data generated by the sensors at $tier\#0$ when the confidence level of the per-tier aggregated data $\vec{\hat{Y}}_m$ delivered by $AG(m)$ is high enough (see the dotted red lines in Fig. 2). In the opposite case (i.e., when the confidence level of $\vec{\hat{Y}}_m$ is estimated to not be high enough), the local output is the empty set: $\emptyset$, that is, no data is generated by the m-th local output;

3. the *uppermost* segment (i.e., $tier\#M$), where a (single) remote Cloud node (labeled as Fog$(1, M)$ in Fig. 2) operates. Its task is to perform complex analytics on the most hard-to-classify input instances, so to provide a final decision $\hat{c}_M$ on the class label of the currently sensed input data, regardless of its actual confidence level.

From time to time, the sensors at the bottom of the technological platform of Fig. 2 sense the local surrounding environment and then pass the sensed data in upstream for its tier-by-tier hierarchical mining. At each intermediate $tier\#m$, with $1 \leq m \leq (M-1)$, an (average) fraction ρ_m of the input data is passed to the next $tier\#(m+1)$ for further processing, while the remaining fraction $(1-\rho_m)$ undergoes local exit, in order to produce the corresponding decision $\hat{c}_m$. Due to the real-time nature of the considered IoT application scenario, the inference-delay at which the decision $\hat{c}_m$ exits at each $tier\#m$, $1 \leq m \leq M$, is assumed to be limited up to a per-tier upper bound $T_{EXIT}^{(m)}$ (s), $1 \leq m \leq M$, whose actual value is set on the basis of the Quality of Service (QoS) requirements of the supported application.

In order to support real-time inference, the time axis, over which the platform of Fig. 2 operates, is partitioned into time-slots which are labeled by a discrete-time slot-index $t \geq 0$. In the case of periodic sensing, the slot duration is fixed at T_S, so that the t-th slot spans the semi-open time-interval: $[tT_S, (t+1)T_S)$. In the more general case of event-driven sensing, the duration

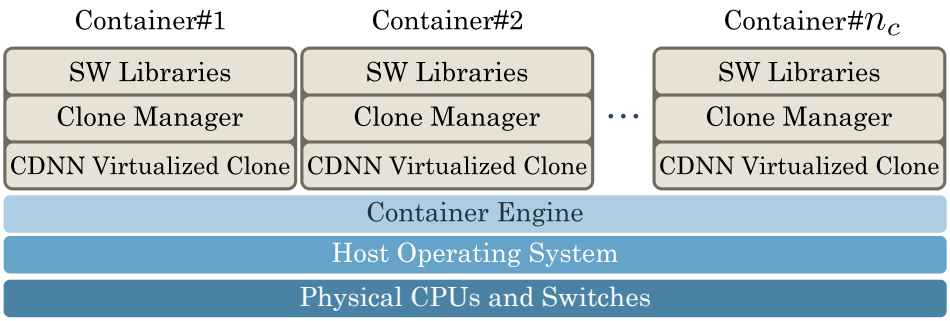


FIGURE 3. Logical view of a virtualized container-based Fog node. n_c is the number of containers hosted by each Fog node.

$\Delta_S(t)$ of the t-th slot depends on the slot-index t, so that the t-th slot covers the semi-open interval: $[\xi(t), \xi(t) + \Delta_S(t))$, where: $\xi(t) \triangleq \sum_{i=0}^{t-1} \Delta_S(i)$ is the starting time of the t-th slot. We anticipate that all the results presented in the sequel hold verbatim in both cases of periodic and event-driven sensing, provided that, in the second case, the minimum slot duration is lower bounded by T_S, that is, $\Delta_S(t) \geq T_S$, $\forall t$.

The envisioned technological platform of Fig. 2 is assumed to be equipped with both networking and computing capabilities. Specifically, the required inter-node message passing is implemented by a number of inter-tier and intra-tier transport connections (see the arrows of Fig. 2). Up-link communication between FNs falling in adjacent tiers is supported by a number of TCP/IP one-way reliable connections, which are sustained by wireless (possibly, single-hop and WiFi-based) communication links (see the continuous purple arrows in Fig. 2 between adjacent tiers). Since load balancing is assumed to be performed at the outputs of each tier, all the transport connections going from Fog nodes at *tier #m* to Fog nodes at *tier #*$(m+1)$ are assumed to operate at a *same* bit-rate of R_m (bit/s). Horizontal communication between the Fog nodes and the Aggregator falling into a same tier is assured by an intra-tier (wired or wireless) LAN which relies on UDP/IP two-way transport connections (see the continuous blue arrows in Fig. 2). Being of local-type and used only for (sporadic) aggregation operations, these intra-tier connections are assumed to operate at low bit-rates, so that the impact of their energy consumption is expected *not* to be so substantial.

1.2 THE CONSIDERED CONTAINER-BASED VIRTUALIZATION OF THE COMPUTING NODES

In principle, the technological platform of Fig. 2 may run in parallel *multiple* CDNNs which carry out different analytics on the same set of sensed data by resorting to the virtualization of the full spectrum of available physical resources [5].

Hence, according to this consideration, we assume that all Fog/Cloud nodes of Fig. 2 are equipped with software clones of the run CDNNs. The number of clones simultaneously hosted by each FN equates to the number of (possibly, multiple) CDNNs which are running in parallel over the technological platform of Fig. 2. So doing, each clone is fully dedicated to the execution of a single associated CDNN, and, then it acts as a virtual "server" by providing resource augmentation to the tied "client" CDNN. For this purpose, each clone is executed by a container (CNT) that is instantiated atop the hosting FN. The container is capable of using (through resource multiplexing) a slice of the physical computing and networking resources of the hosting FN. The logical view of the resulting virtualized container-based FN is detailed in Fig. 3.

All containers hosted by Fog(j,m) in Fig. 3 share: (i) a same Host Operating System (HOS); and, (ii) the pool of computing (i.e., CPU cycles) and networking (i.e., bandwidth and I/O Network Interface Cards (NICs) and switches) physical resources available on the hosting FN. The task of the Container Engine of Fig. 3 is to allocate these physical resources to the requiring containers by performing dynamical multiplexing.

The resulting virtualized architecture of Fog(j,m) is sketched in Fig. 4. Specifically, in Fig. 4 we have that:

1. the (j,m)-th *Virtualized Convolutional Processor*, running at frequency f_{jm}, provides the computing support for the execution of the set of $|\mathcal{T}_m|$ consecutive convolutional and pooling layers of the supported CDNN to be executed on *tier #m*. In the envisioned architecture, the input data $\vec{z}_{m-1}(j)$ at the *Virtualized Convolutional Processor* is a feature vector received from the FNs working at the previous *tier #*$(m-1)$, while the corresponding output data $\vec{z}_m(j)$ is provided to the associated *Virtualized Classifier Processor*, as well as forwarded to the FNs operating at the next *tier #*$(m+1)$;

2. the (j,m)-th *Virtualized Classifier Processor*, running at frequency $\widetilde{f}_{jm}$, provides the computing support for the execution of the local classifier of the CDNN. Its task is to process the feature vector generated by the corresponding *Virtualized Convolutional Processor*, so to produce the output vector $\vec{\hat{y}}_m(j)$ to be delivered to the corresponding m-th *Local Aggregator* of Fig. 2 for the (possible) generation of an early-exit;

3. Fog(j,m) is also equipped with a number $FanIn(j,m) \geq 1$ of virtualized input ports. Hence, the task of the *MUltipleXer* (*MUX*) at the bottom of Fig. 4 is to merge the corresponding information flows received by the FNs operating at the previous *tier*#$(m-1)$, so to generate a (single) aggregate feature vector $\vec{z}_{m-1}(j)$. All input flows at the bottom of Fig. 4 are assumed to operate at a same bit-rate R_{m-1} (bit/s);

4. the main task of the *De-MUltipleXer* (*DEMUX*) at the top of Fig. 4 is to replicate the received feature vector $\vec{z}_m(j)$ over the $FanOut(j,m) \geq 1$ virtualized output ports equipping Fog(j,m). Each output flow is forwarded to the FNs at the next *tier*#$(m+1)$ at a bit-rate R_m (bit/s);

5. the task of the *Virtual Switch* (*VS*) of Fig. 4 is to enable the transmission of the output feature $\vec{z}_m(j)$ to the FNs at the next *tier*#$(m+1)$ only when early-exit does not occur at *tier*#m.

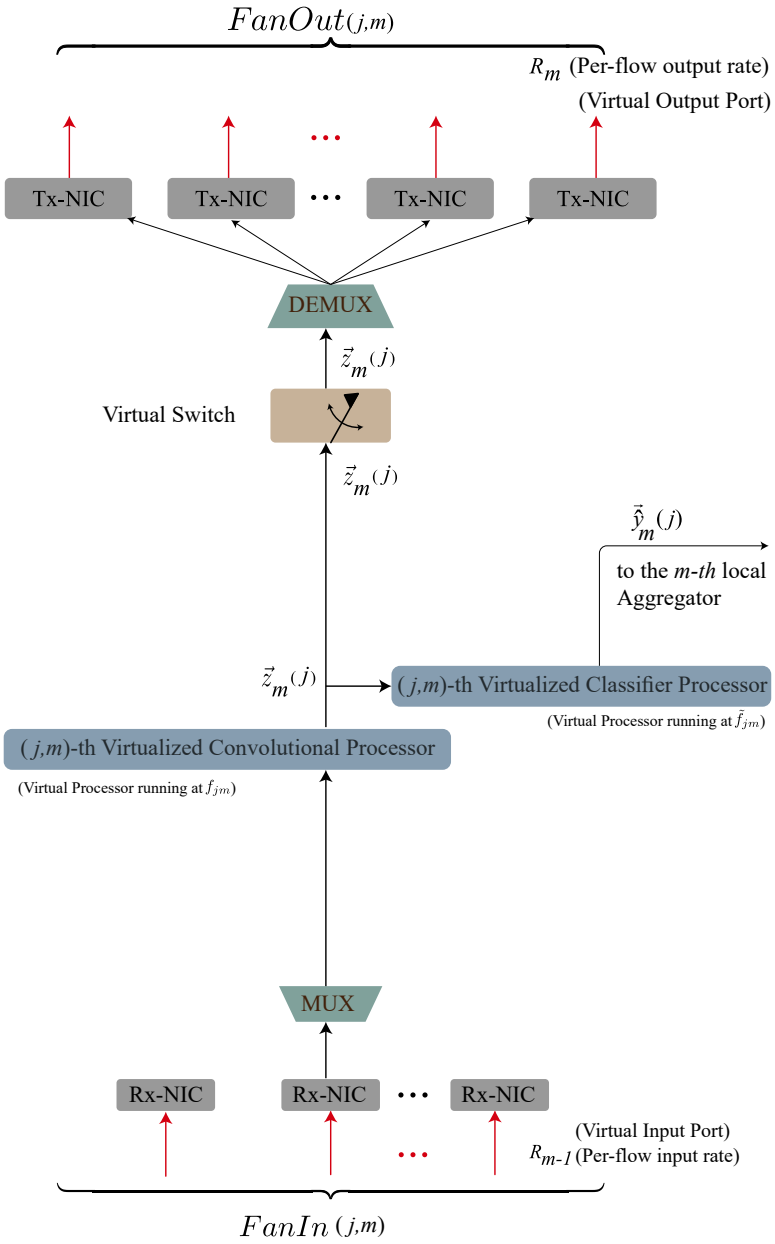


FIGURE 4. Envisioned virtualized architecture of Fog(j,m). It is hosted by a container and implemented by the associated virtualized clone. NIC:= Network Interface Card; MUX:= Multiplexer; DEMUX:= De-multiplexer.

2 DEEPFOGSIM – THE UNDERLYING OPTIMIZATION PROBLEM

In this section, the basic definitions and formal assumptions of the constrained optimization problem are shortly explained.

Since all FNs of a same tier of Fig. 2 cooperate, by definition, in performing the same type of processing, all flows output by FNs of *tier #m* have the same throughput R_m (bit/s), with $1 \leq m \leq M-1$.

In the case that $\left|\mathcal{T}_m\right|$ consecutive layers of the supported CDNN collapse into a single component layer at the m-th tier, the corresponding layer interior architecture becomes composed of the cascade of $\left|\mathcal{T}_m\right|$ convolutional blocks followed by a single nonlinearity and a single soft-decisor that sits atop the last convolutional blocks. As a consequence of these architecture features, we have that:

1. the processing density of each (j,m)-th Fog *convolutional processor* passes from ε_m to $\left|\mathcal{T}_m\right|\varepsilon_m$;
2. the processing density of each (j,m)-th Fog *classifier processor* is β_m, regardless of the number $\left|\mathcal{T}_m\right|$ of CDNN layers, which are executed by the FNs at *tier #m*;
3. the average volume of data $v_I(j,m)$ (bit) received in input by Fog(j,m) during each sensing period is:

$$v_I(j,m) \triangleq E\left\{dim\left(\vec{z}_{m-1}(j)\right)\right\} = v_0 \times \left(\prod_{k=1}^{\sum_{j=1}^{m-1}|\mathcal{T}_j|} cm_k\right) \times \frac{FanIn(j,m)}{\sum_{\tau=1}^{m_m} FanIn(\tau,m)}, \tag{1}$$

   where $E\{\cdot\}$ denotes the expectation and $dim\left(\vec{z}\right)$ is the size (in bit) of the vector $\vec{z}$;
4. the average volume of data $v_O(j,m)$ (bit) output by Fog(j,m) is:

$$v_O(j,m) \triangleq E\left\{dim\left(\vec{z}_m(j)\right)\right\} = v_0 \times \left(\prod_{k=1}^{\sum_{j=1}^{m}|\mathcal{T}_j|} cm_k\right) \times \frac{FanIn(j,m)}{\sum_{\tau=1}^{m_m} FanIn(\tau,m)}. \tag{2}$$

2.1 THE CONSIDERED JOINT COMPUTING-PLUS-NETOWRKING RESOURCES ALLOCATION PROBLEM

From a formal point of view, the proposed *DeepFogSim* toolkit numerically computes the solution and evaluates the resulting performance of an optimization problem that regards the *joint* allocation of the available computing-plus-networking resources to the FNs of the virtualized execution platform of Fig. 2 under per-exit constraints on the tolerated inference delays.

Let $\mathcal{E}_{TOT}$ (Joule) be the (average) overall energy wasted by the virtualized networked computing platform of Fig. 2 over a (single) sensing period for both network and computing operations. Hence, by definition, we have that:

$$\mathcal{E}_{TOT} \equiv \mathcal{E}_{COP} + \mathcal{E}_{NET}, \tag{3}$$

where the overall computing energy $\mathcal{E}_{COP}$ equates to (see Fig. 4):

$$\mathcal{E}_{COP} = \sum_{m=1}^{M}\sum_{j=1}^{m_m}\left(\mathcal{E}_{CON}(j,m) + \mathcal{E}_{CLA}(j,m)\right), \tag{4}$$

while the overall network energy $\mathcal{E}_{NET}$ is given by:

$$\mathcal{E}_{NET} = \sum_{j=1}^{m_1}\mathcal{E}_{NET}^{Tx}(j,1) + \sum_{m=2}^{M-2}\sum_{j=1}^{m_m}\left(\mathcal{E}_{NET}^{(Rx)}(j,m) + \mathcal{E}_{NET}^{(Tx)}(j,m)\right) + \mathcal{E}_{NET}^{(Rx)}(1,M). \tag{5}$$

Eq. (5) is the summations of three terms, i.e.: (i) the energy wasted by *tier #1* for transmission; (ii) the energy consumed by all intermediate tiers for receive and network operations; and, (iii) the energy consumed by the uppermost Cloud node for receiving purposes.

Before proceeding, a remark about the formal meaning of $\mathcal{E}_{TOT}$ is in order. Specifically, according to the virtualized nature of the proposed *DeepFog* technological platform of Fig. 3, $\mathcal{E}_{TOT}$ in (3) is the total (i.e., computing-plus-communication) energy wasted over a (single) sensing period by *all and only* Fog clones that actually support the execution of the considered CDNN. Hence, $\mathcal{E}_{TOT}$ considers *only* the energy consumed for the execution of the supported CDNN of Fig. 1a and does *not* represent the total energy wasted by the full hardware infrastructure of Fig. 1b, which can be used for the simultaneous support of *multiple*

jobs and/or CDNNs. This meaning of $\mathcal{E}_{TOT}$ is in agreement with the virtualized architecture of each FN that has been previously detailed in Figs. 3 and 4.

In order to introduce this problem, let $Q = \sum_{m=1}^{M} m_m$ be the number of FNs of the platform of Fig. 2. Let: $\vec{f} \triangleq \left[f_{11}, \ldots, f_{M1}\right]^T \in \mathbb{R}_+^Q$, $\overrightarrow{\widetilde{f}} \triangleq \left[\widetilde{f}_{11}, \ldots, \widetilde{f}_{M1}\right]^T \in \mathbb{R}_+^Q$ and $\vec{R} \triangleq \left[R_1, \ldots, R_{M-1}\right]^T \in \mathbb{R}_+^{M-1}$ be the vectors of the convolutional and classifier processing frequencies and the vector of the inter-tier network throughputs, respectively. Finally, let $\vec{x} \triangleq \left[\vec{f}, \overrightarrow{\widetilde{f}}, \vec{R}\right]^T \in \mathbb{R}_+^{2Q+M-1}$ be the resulting compound vector. Hence, the tackled constrained resource allocation problem is formally defined as follows:

$$\min_{\vec{x}} \mathcal{E}_{TOT}\,, \tag{6a}$$

s.t.:

$$0 \leq T_{EXE}^{(1,m)} \leq T_{EXIT}^{(m)}\,, \qquad m = 1, \ldots, M\,, \tag{6b}$$

$$0 \leq f_{jm} \leq f_{jm}^{(MAX)}\,, \qquad j = 1, \ldots, m_m\,, \quad m = 1, \ldots, M\,, \tag{6c}$$

$$0 \leq \widetilde{f}_{jm} \leq \widetilde{f}_{jm}^{(MAX)}\,, \qquad j = 1, \ldots, m_m\,, \quad m = 1, \ldots, M\,, \tag{6d}$$

$$0 \leq R_m \leq R_m^{(MAX)}\,, \qquad m = 1, \ldots, M\,, \tag{6e}$$

where: $\left\{T_{EXIT}^{(m)},\ m = 1, \ldots, M\right\}$ (s), $\left\{f_{jm}^{(MAX)},\ j = 1, \ldots, m_m,\ m = 1, \ldots, M\right\}$ (bit/s), $\left\{\widetilde{f}_{jm}^{(MAX)},\ j = 1, \ldots, m_m,\ m = 1, \ldots, M\right\}$ (bit/s) and $\left\{R_m,\ m = 1, \ldots, M-1\right\}$ (bit/s), are the set of the $M + 2Q + M - 1 = 2(M+Q) - 1$ assigned constraints on the maximum allowed resources and tolerated per-exit inference delays.

The box constraints in (6c)–(6e) upper bound the computing frequencies at the FN and the inter-tier network throughput, respectively. The M constraints in (6b) guarantee that each local exit is generated within a maximum inference delay. In the case in which no delay constraint is enforced to the m-th local exit, we set $T_{EXIT}^{(m)} = +\infty$.

The problem in (6) is a continuous-valued optimization problem that embraces $(2Q + M - 1)$ non-negative continuous optimization variables and $2(M + Q) - 1$ constraints. The $(2Q + M - 1)$ constraints in (6c)–(6e) are of box-type, while the M delay constraints in (6b) are nonlinear constraints involving the optimization vector $\vec{x}$.

2.2 SIMULATED PROFILES OF THE PER-NODE AND PER-TIER EXECUTION TIME

Since we assume that the transmit Network Interface Cards (NICs) of Fog(j, m), at the top of Fig. 4, may operate in parallel, the per-node execution time $T_{EXE}(j, m)$ can be expressed as the summation of three terms, i.e., the computation time $T_{CON}(j, m)$, the classification time $T_{CLA}(j, m)$ and the network transmission time $T_{NET}(j, m)$. Hence, it equates:

$$T_{EXE}(j, m) \triangleq T_{CON}(j, m) + T_{CLA}(j, m) + T_{NET}(j, m) \equiv \frac{v_I(j, m)}{f_{j,m}} + \frac{v_O(j, m)}{\widetilde{f}_{j,m}} + \frac{v_I(j, m)}{R_m}, \tag{7}$$

where f_{jm} (resp., $\widetilde{f}_{jm}$) is the processing speed (in (bit/s)) of the convolutional processor (resp., the classifier processor) equipping Fog(j, m) (see Fig. 4).

Furthermore, since the Cloud does not perform transmission but only convolutional and classify tasks, its execution time comprises only the first two terms of the above expression, that is:

$$T_{EXE}(1, M) \triangleq T_{EXE}^{(Cloud)} \equiv \frac{v_I(1, M)}{f_{1,M}} + \frac{v_O(1, M)}{\widetilde{f}_{1,M}}. \tag{8}$$

Since the m-th local aggregator in Fig. 2 must receive all the $|\mathcal{Q}|$-dimensional vectors $\left\{\vec{y}_j(m) \in [0,\ 1]^{|\mathcal{Q}|},\ 1 \leq j \leq m_m\right\}$ before performing its final operation, the (per-tier) execution time $T_{EXE}(m)$ at *tier*#m equates to the execution time of the *slowest* FN, so that we have:

$$T_{EXE}(m) \equiv \max_{1 \leq j \leq m_m} \left\{T_{EXE}(j, m)\right\}. \tag{9}$$

By definition, the multi-tier Fog architecture of Fig. 2 processes the data generated by IoT devices of Fig. 2 in a *sequential* way. Hence, the aggregate execution time $T_{EXE}^{(1,m)}$ needed to generate the local exit at $tier\,\#m$ equates to:

$$T_{EXE}^{(1,m)} = \sum_{k=1}^{m} T_{EXE}(k) \equiv \sum_{k=1}^{m} \max_{1\leq j\leq m_k} \left\{T_{EXE}(j,k)\right\}, \quad 1 \leq m \leq M, \tag{10}$$

so that, the resulting time constraint on the allowed local decision at $tier\,\#m$ reads as follows:

$$T_{EXE}^{(1,m)} \leq T_{EXIT}^{(m)}, \quad 1 \leq m \leq M, \tag{11}$$

where $T_{EXIT}^{(m)}$ is the tolerated upper bound on the time needed for the generation of the local decision at $tier\,\#m$.

2.3 SIMULATED PROFILES OF THE COMPUTING AND NETWORK ENERGY

In this subsection, we provide the formal relationships modeling the power and energy consumption of each FN involved by the objective function in (6a). Specifically, we analyze separately the contributions of the convolutional and classifier processors equipping each FN in Fig. 4. The goal is to allow the *DeepFogSim* user to acquire a basic insight about the played roles and possible impacts of the input parameters in Table A of the simulator on the resulting solution $\vec{x}$ of the underlying optimization problem in (6).

2.3.1 Models of the per-node computing power and energy

The power wasted by the convolutional processor of Fog(j,m) in Fig. 4 is composed of a static part $\mathcal{P}_{CON}^{(IDLE)}(j,m)$ (Watt), with $1 \leq j \leq m_m$ and $1 \leq m \leq M$, and a dynamic one:

$$\mathcal{P}_{CON}^{(DYN)}(j,m) = K_{CON}^{(j,m)} \left(\left|\mathcal{T}_m\right| \varepsilon_{jm}\, f_{ij}\right)^{\gamma_{CON}^{(j,m)}}, \quad 1 \leq j \leq m_m,\ 1 \leq m \leq M, \tag{12}$$

where [6]:

1. $\gamma_{CON}^{(j,m)}$ is a dimensionless power exponent that depends on the power profile of the underlying CPU;
2. ε_{jm} (CPU cycle/bit) is the so called processing density of the computation operations performed by the underlying CPU when a single layer of the CDNN must be executed. It depends on the number of convolutional neurons that compose a single layer of the considered CDNN and the average number of summations/multiplications performed by each neuron [2];
3. f_{jm} (bit/s) is the processing frequency of the (j,m)-th convolutional processor;
4. $\left|\mathcal{T}_m\right|$ is the number of layers of the underlying CDNN to be executed by Fog(j,m). This term accounts for the fact that the workload to be processed by the (j,m)-th convolutional processor scales (more or less) linearly with the number $\left|\mathcal{T}_m\right|$ of convolutional layers of CDNN to be executed by the Fog(j,m) [2]; and,
5. $K_{CON}^{(j,m)}$, measured in (Watt/(CPU cycle/s)$^{\gamma_{CON}^{(j,m)}}$), is a power scaling factor that depends on the power profile of the CPU supporting the (j,m)-th convolutional processor.

In order to compute the (average) energy $\mathcal{E}_{CON}^{(j,m)}$ (Joule) consumed by the (j,m)-th convoultional processor during each sensing interval, we note that: (i) the (j,m)-th convolutional processor must remain turned ON for a time $T_{EXE}^{(1,M)}$ equal to the execution time required by the Cloud node of Fig. 2, i.e., the *maximum* time needed for the processing of the data generated by the server in a sensing period; and, (ii) since the (j,m)-th convolutional processor works at the processing speed f_{jm} (bit/s) and the workload to be executed is $v_I(j,m)$ in (1), the resulting processing time is $T_{CON}(j,m)$ in (7). Hence, we have the following relationship for $\mathcal{E}_{CON}(j,m)$:

$$\begin{aligned}\mathcal{E}_{CON}(j,m) &\triangleq \mathcal{E}_{CON}^{(IDLE)}(j,m) + \mathcal{E}_{CON}^{(DYN)}(j,m) \equiv \mathcal{P}_{CON}^{(IDLE)}(j,m) \times T_{EXE}^{(1,M)} + \mathcal{P}_{CON}^{(DYN)}(j,m) \times T_{CON}(j,m) \\ &= \mathcal{P}_{CON}^{(IDLE)}(j,m) \times T_{EXE}^{(1,M)} + v_I(j,m) K_{CON}^{(j,m)} \left(\left|\mathcal{T}_m\right| \varepsilon_{jm}\right)^{\gamma_{CON}^{(j,m)}} \times f_{ij}^{\gamma_{CON}^{(j,m)}-1},\end{aligned} \tag{13}$$

where $T_{EXE}^{(1,M)} \equiv \sum_{k=1}^{M} \max_{1\leq j\leq m_k} \left\{T_{EXE}(j,k)\right\}$ is the full (i.e, worst case) execution time of the supported CDNN of Fig. 1a.

Let $\mathcal{P}_{CLA}^{(IDLE)}(j,m)$ (Watt) be the idle power consumed by the CPU of the (j,m)-th classifying processor in Fig. 4. Then, the corresponding dynamic power component may be modeled as [6]:

$$\mathcal{P}_{CLA}^{(DYN)}(j,m) = K_{CLA}^{(j,m)} \left(\beta_{jm}\, \widetilde{f}_{jm}\right)^{\gamma_{CLA}^{(j,m)}}, \quad 1 \le j \le m_m,\ 1 \le m \le M, \tag{14}$$

where:

1. $\gamma_{CLA}^{(j,m)}$ is a dimensionless power exponent that depend on the power profile of the underlying CPU;
2. $\widetilde{f}_{jm}$ (bit/s) is the processing frequency of the (j,m)-th classifying processor;
3. β_{jm} (CPU cycle/bit) is the processing density of the computation operations performed by the underlying CPU [2];
4. $K_{CLA}^{(j,m)}$, measured in (Watt/(CPU cycle/s)$^{\gamma_{CLA}^{(j,m)}}$), is a positive scaling factor that depends on the power consumption of the CPU that supports the (j,m)-th classifier.

Hence, the energy $\mathcal{E}_{CLA}(j,m)$ (Joule) consumed by the (j,m)-th classifier during a sensing interval equates to:

$$\begin{aligned}\mathcal{E}_{CLA}(j,m) &\triangleq \mathcal{E}_{CLA}^{(IDLE)}(j,m) + \mathcal{E}_{CLA}^{(DYN)}(j,m) \equiv \mathcal{P}_{CLA}^{(IDLE)}(j,m) \times T_{EXE}^{(1,M)} + \mathcal{P}_{CLA}^{(DYN)}(j,m) \times T_{CLA}(j,m) \\ &= \mathcal{P}_{CLA}^{(IDLE)}(j,m) \times T_{EXE}^{(1,M)} + v_O(j,m) K_{CLA}^{(j,m)} \times \left(\beta_{jm}\right)^{\gamma_{CLA}^{(j,m)}} \times \left(f_{ij}\right)^{\gamma_{CLA}^{(j,m)}-1},\end{aligned} \tag{15}$$

with $T_{EXE}^{(1,M)}$ being still the full (i.e, worst case) execution time.

2.3.2 Models of the per-flow networking power and energy

Before developing the power/energy network models, some remarks about the network aspects of the Fog platform of Fig. 2 are in order.

First, we assume that the m-th bit-rate R_m (bit/s), $1 \le m \le M-1$ in Fig. 2 is a per-flow transport-layer throughput. However, both the consumed network power and energy depend on the corresponding bit-rate $\widehat{R}_m$ measured at the physical layer. In general, the (average values of the) above communication rates may be related as:

$$\widehat{R}_m = \psi_m R_m, \quad 1 \le m \le M-1, \tag{16}$$

where ψ_m is a dimensionless coefficient that accounts for the communication overhead incurred by passing from the Transport layer to the Physical one of the underlying protocol stack. Typical values of ψ_m are in the range: $1.45 \le \psi_m \le 1.70$ [5].

Second, each FN Fog(j,m) must act as both a computing and a (wireless) network switch. Furthermore, depending on the actual network topology, in general, its fan-in and fan-out may be different, i.e., $FanIn(j,m) \lesseqgtr FanOut(j,m)$. According to this consideration, Fog(j,m) in Fig. 4 is equipped with two distinct sets of input and output NICs.

Third, since all FNs that compose the same tier cooperate in the execution of the same set of CDNN layers, it is reasonable to retain that the total volume of data $v_I(j,m)$ received in input by Fog(j,m) is *balanced* over its $FanIn(j,m)$ input ports.

The mentioned features of the networking infrastructure of the envisioned Fog execution platform of Fig. 2 will be exploited in the sequel for developing the corresponding network power and energy models.

Let $\mathcal{P}_{NET}^{(IDLE;\,Rx)}(j,m)$ (Watt) be the idle power consumed by each receive port of Fog(j,m) in Fig. 4. Hence, the corresponding receive dynamic power can be modeled as [6]:

$$\mathcal{P}_{NET}^{(DYN;\,Rx)}(j,m) = \Omega_{NET}^{(Rx)}(j,m) \left(\psi_m R_m\right)^{\zeta^{(Rx)}(j,m)}, \quad 2 \le m \le M. \tag{17}$$

Now, the idle time is still $T_{EXE}^{(1,M)}$. However, the receivie time is:

$$T_{NET}^{(Rx)}(j,m) = \frac{v_I(j,m)}{R_{m-1} \times FanIn(j,m)}, \quad 2 \le m \le M, \tag{18}$$

because the overall workload $v_I(j,m)$ received by Fog(j,m) is evenly split over its $FanIn(j,m)$ input ports, each one working at R_{m-1}. Hence, the total network energy $\mathcal{E}_{NET}^{(Rx)}(j,m)$ consumed by Fog(j,m) for receiving operations over a sensing interval

equates to:

$$\begin{aligned}\mathcal{E}_{NET}^{(Rx)}(j,m) &\triangleq \mathcal{E}_{NET}^{(IDLE;\,Rx)}(j,m) + \mathcal{E}_{NET}^{(DYN;\,Rx)}(j,m)\\ &\equiv FanIn(j,m) \times \Big\{ \mathcal{P}_{NET}^{(IDLE;\,Rx)}(j,m) \times T_{EXE}^{(1,M)}\\ &\quad + \Omega_{NET}^{(Rx)}(j,m) \left(\psi_{m-1} R_{m-1}\right)^{\zeta^{(Rx)}(j,m)} T_{NET}^{(Rx)}(j,m) \Big\}\\ &= FanIn(j,m) \times \mathcal{P}_{NET}^{(IDLE;\,Rx)}(j,m) \times T_{EXE}^{(1,M)} + \upsilon_I(j,m) \times \Omega_{NET}^{(Rx)}(j,m)\\ &\quad \times \left(\psi_{m-1}\right)^{\zeta^{(Rx)}(j,m)} \times \left(R_{m-1}\right)^{\zeta^{(Rx)}(j,m)-1}.\end{aligned} \tag{19}$$

The above expression holds for $m \geq 2$ because, in our framework, sensors are co-located with FNs at *tier*#1 (see Fig. 2). $\Omega_{NET}^{(Rx)}(j,m)$ and $\zeta^{(Rx)}(j,m)$ in (19) depend on the communication technology employed at the Physical layer and can be modeled as follows [6]:

- $\zeta^{(Rx)}(j,m) \geq 2$ is a positive dimension-less exponent, whose actual value depends on the transmission technology actually implemented at the Access sub-layer of the functional stack [7, 8];
- $\Omega_{NET}^{(Rx)}(j,m)$ is a positive coefficient (measured in Watt/((bit/s)$^{\zeta}$ $\times$ (s)$^{\eta}$)), which accounts for the effect of the round-trip-time RTT_{m-1} of the received flows. Specifically, in the case of single-hop connections, it can be formally modeled as in the following [6, 9]:

$$\Omega_{NET}^{(Rx)}(j,m) = \frac{\left(RTT_{m-1}\right)^{\eta} \chi(j,m)^{(Rx)}}{1 + \left(l_{m-1}\right)^{\alpha}}, \tag{20}$$

  where: (i) $\eta \approx 0.6$ is a dimension-less positive exponent; (ii) l_{m-1} is the length (i.e., coverage, measured in meter (m)) of the wireless connections in Fig. 2 going from *tier*#$(m-1)$ to *tier*#m; (iii) $\alpha \geq 2$ is a fading-induced path-loss exponent; and, (iv) the positive coefficient $\chi(j,m)^{(Rx)}$ accounts for the power profile of the receive ports.

Let $\mathcal{P}_{NET}^{(IDLE;\,Tx)}(j,m)$ (Watt) be the idle power consumed by a single transmit port of Fog(j,m). Hence, the corresponding per-port dynamic transmit power can be modeled as [6]:

$$\mathcal{P}_{NET}^{(DYN;\,Tx)}(j,m) = \Omega_{NET}^{(Tx)}(j,m) \left(\psi_m R_m\right)^{\zeta^{(Tx)}(j,m)}, \quad 1 \leq m \leq M-1. \tag{21}$$

Now, the idle time is still $T_{EXE}^{(1,M)}$. However, since the overall data $\upsilon_0(j,m)$ generated by Fog(j,m) is entirely replicated over each output port (see Fig. 4), the corresponding transmit time equates to:

$$T_{NET}^{(Tx)}(j,m) = \frac{\upsilon_0(j,m)}{R_m}. \tag{22}$$

Hence, the total energy $\mathcal{E}_{NET}^{(Tx)}(j,m)$ wasted by Fog(j,m) over a sensing interval for transmission purpose can be modeled as follows:

$$\begin{aligned}\mathcal{E}_{NET}^{(Tx)}(j,m) &\triangleq \mathcal{E}_{NET}^{(IDLE;\,Tx)}(j,m) + \mathcal{E}_{NET}^{(DYN;\,Tx)}(j,m)\\ &\equiv FanOut(j,m) \times \Big\{ \mathcal{P}_{NET}^{(IDLE;\,Tx)}(j,m) \times T_{EXE}^{(1,M)}\\ &\quad + \Omega_{NET}^{(Tx)}(j,m) \left(\psi_m R_m\right)^{\zeta^{(Tx)}(j,m)} \times T_{NET}^{(Tx)}(j,m) \Big\}\\ &= FanOut(j,m) \times \Big\{ \mathcal{P}_{NET}^{(IDLE;\,Tx)}(j,m) \times T_{EXE}^{(1,M)} + \upsilon_O(j,m) \times \Omega_{NET}^{(Tx)}(j,m)\\ &\quad \times \left(\psi_m\right)^{\zeta^{(Tx)}(j,m)} \times \left(R_m\right)^{\zeta^{(Tx)}(j,m)-1} \Big\}.\end{aligned} \tag{23}$$

The above expression holds for $1 \leq m \leq M-1$ because, by design, the last node (i.e., the Cloud) in Fig. 2 does not transmit. The parameters $\zeta^{(Tx)}(j,m)$ and $\Omega_{NET}^{(Tx)}(j,m)$ in Eq. (23) play the same roles and, then, assume the same formal expressions to the receive counterparts in Eqs. (19) and (20).

2.4 SIMULATED ADAPTIVE RESOURCE ALLOCATION

After indicating by $\mathcal{L}$ the Lagrangian function of the problem in Eq. (6), and by $\{\lambda_m,\ m = 1, \dots, M\}$ the Lagrange multipliers associated to the delay constraints in (6b), let: $\{\vec{x}^*,\ \vec{\lambda}^*\}$ be a solution of the optimization problem. In order to compute it, we implement the iteration-based primal-dual approach recently customized in [10] for broadband networked application scenarios. The primal-dual algorithm is an iterative procedure that updates on per-step basis both the primal $\vec{x}$ and the dual $\vec{\lambda}$ variables, in order to throttle the corresponding Lagrangian function $\mathcal{L}$ towards its saddle point. Hence, after introducing the dummy position $[z]_a^b \triangleq \max\{a, \min\{z, b\}\}$, the $(k+1)$-th update of the i-th scalar component x_i, $i = 1, \dots, (2Q + M - 1)$ of the resource vector $\vec{x}$ reads as:

$$x_i^{(k+1)} = \left[x_i^{(k)} - \alpha_i^{(k)} \frac{\partial \mathcal{L}\left(\vec{x}^{(k)}, \vec{\lambda}^{(k)}\right)}{\partial x_i} \right]_0^{x_i}, \quad k \geq 0, \quad i = 1, \dots, 2Q + M - 1, \tag{24}$$

while the $(k+1)$-th update of the m-th scalar component of the Lagrnage multiplier vector $\vec{\lambda}$ is:

$$\lambda_m^{(k+1)} = \left[\lambda_m^{(k)} - \xi_m^{(k)} \frac{\partial \mathcal{L}\left(\vec{x}^{(k)}, \vec{\lambda}^{(k)}\right)}{\partial \lambda_m} \right]_0^{+\infty}, \quad k \geq 0, \quad m = 1, \dots, M. \tag{25}$$

In order to guarantee fast responses to abrupt (and possibly unpredictable) changes of the operating conditions as well as (asymptotic) convergence to the steady state of the iterations in (24) and (25), we implemented the following "clipped" relationships for updating the step sizes in (24) and (25):

$$\alpha_i^{(k)} = \max\left\{ a_{MAX}; \min\left\{ a_{MAX} \times \left(x_i^{(MAX)}\right)^2; \left(x_i^{(k)}\right)^2 \right\} \right\}, \quad i = 1, \dots, 2Q + M - 1, \tag{26}$$

and

$$\xi_i^{(k)} = \max\left\{ a_{MAX}; \min\left\{ a_{MAX} \times \max_i\left\{ \left(x_i^{(MAX)}\right)^2 \right\}; \left(\lambda_m^{(k)}\right)^2 \right\} \right\}, \quad m = 1, \dots, M. \tag{27}$$

The goal of the clipping factor a_{MAX} is to avoid both too small and too large values of the step-size in (24) and (25), in order to guarantee a quick self-response to operative changes and small oscillations in the steady state.

3 DEEPFOGSIM – THE BUILT-UP RESOURCE ALLOCATION STRATEGIES

The core of the *DeepFogSim* simulator is built up by three software routines that implement a number of strategies (i.e., optimization policies), in order to numerically solve the constrained optimization problem in (6). MATLAB is the native environment under which the optimization routines are developed and run. Differently from our previous simulator [11], the problem solved here is intrinsically sequential, hence the current version of *DeepFogSim* does not exploit the parallel programming on multi-core hardware platforms supported by the *Parallel Tool Box* of MATLAB, which is utilized by the *VirtFogSim* package.

The main functionalities of *DeepFogSim* are implemented by the three functions: *DynDeF_RAS*, *Static_Allocation*, and *DynDeFog_TRACKER*, which will be described in Section 3. These main functions use some auxiliary functionalities implemented by the set of additional routines described in Section 3.6.

To describe the general software architecture of the simulator, we note that *DeepFogSim* acts as the main program that:

1. allows the user to setup 34 input parameters (refer to Table A), that characterize the scenario to be simulated by the user (see Fig. 2);
2. calls the *DynDeF_RAS* function to run the Resource Allocation Strategy to solve the optimization problem in Eq. (6) (see Section 3.3);
3. optionally calls the *Static_Allocation* function to simulate the same scenario where all the resources are set to their maximum values $\left\{ \vec{f}^{(MAX)},\ \vec{\vec{f}}^{(MAX)},\ \vec{R}^{(MAX)} \right\}$ (bit/s) (see Section 3.4);

4. optionally, calls the *DynDeFog_TRACKER* function (see Section 3.5). It returns the time plots over the settable interval: [1, *iteration_number*] of the:

   (a) total energy $\mathcal{E}_{TOT}$ consumed by the overall proposed platform of Fig. 2;

   (b) the corresponding energy $\mathcal{E}_{COM}$ consumed by the convolutional and classifier processors of Fig. 2;

   (c) the corresponding energy $\mathcal{E}_{NET}$ consumed by the network connections of Fig. 2; and,

   (d) the behavior of the first and last (M-th) *lambda* multipliers in (25) associated to the constraints in Eq. (6),

   when unpredicted and abrupt changes in the operating conditions of the simulated Fog platform of Fig. 2 occur. The user may set the magnitude of these changes, in order to test various time-fluctuations of the simulated environment of Fig. 2 (see Section 3.5 for a deeper description of the *DynDeFog_TRACKER* function and its supported options).

Moreover, the current version of the *DeepFogSim* simulator is equipped with a (rich) *Graphical User Interface* (GUI) that displays:

1. the numerical values of the best optimized frequencies for the convolutional and classifier processors and the network throughputs; and,

2. the numerical values of the optimal energy consumption, which are returned by the *DynDeF_RAS* function.

3.1 THE INPUT PARAMETERS TO THE DEEPFOGSIM SIMULATOR

DeepFogSim exposes a GUI which allows the user to:

i. set 34 input parameters, in order to define the desired computing and communication setup of the overall infrastructure of Fig. 2;

ii. select any subset of the (aforementioned) *DynDeF_RAS*, *Static_Allocation*, and *DynDeFog_TRACKER* functions, in order to test and compare various resource allocation strategies over different topologies and scenarios dictated by the desired input parameters.

In the sequel, the input parameters of the *DeepFogSim* simulator are listed, together with their meaning, measuring units and ranges of likelihood values.

TABLE A. Input parameters of the *DeepFogSim* simulator version 4.0.

INPUT PARAMETERS	MEANING/ROLE	MEASURING UNITS	SIMULATED SETTINGS
L	Number of the CDNN layers	Dimensionless	$L = 9$
$\vec{cm}$	Vector of per-layer compression	Dimensionless	$0 < cm(m) \leq 1$
T_S	Duration of an inter-sensing period	(s)	$T_S = 1$
M	Number of tiers of the considered Fog platform	Dimensionless	$3 \leq M \leq 6$
$\overrightarrow{pertier}_{nodes}$	Vector collecting per-tier numbers of Fog nodes	Dimensionless	$1 \leq pertier_{nodes}(m) \leq 32$
Q	Total number of Fog nodes	Dimensionless	$7 \leq Q \leq 63$
$\vec{R}^{(MAX)}$	Vector of maximum transport rate of each directed UDP/IP transport connection	(Mbit/s)	$8 \leq R^{(MAX)}(m) \leq 9$
$\vec{v}_0$	Vector of volumes of input data	(Mbit)	$1.5 \leq v_0(m) \leq 1.5$
$[A]$	Matrix describing the topology of the simulated multi-tier Fog platform	Dimensionless	Matrix entries in set $\{0, 1\}$
$\vec{\varepsilon}$	Vector of per-node processing densities of convolutional processor	(CPU cycles/bit)	$10^3 \leq \varepsilon(m) \leq 5 \times 10^3$
$\vec{\beta}$	Vector of per-node processing densities of classifiers	(CPU cycles/bit)	$0.5 \times 10^3 \leq \beta(m) \leq 2.5 \times 10^3$
$\overrightarrow{L2T}_{map}$	Vector collecting the number of the CDNN layers that are mapped onto the m-th Fog tier, $m = 1, \ldots, M$	Dimensionless	$1 \leq L2T_{map}(m) \leq 4$

TABLE A (Continued).

INPUT PARAMETERS	MEANING/ROLE	MEASURING UNITS	VALUE RANGE
$\vec{T}_{EXIT}$	Vector of the maximum tolerated delays for the per-tier local exits	(s)	$0.5 \leq T_{EXIT}(m) \leq T_S$
$\vec{K}_{CON}$	Vector of the scaling coefficients of the dynamic power consumed by the convolutional processors	(Watt/(CPU cycles/s)$^{\gamma_{CON}}$)	$0 \leq K_{CON} \leq 5 \times 10^{-36}$
$\vec{K}_{CLA}$	Vector of the scaling coefficients of the dynamic power consumed by the classifier processors	(Watt/(CPU cycles/s)$^{\gamma_{CLA}}$)	$0 \leq K_{CLA} \leq 5 \times 10^{-36}$
$\vec{\gamma}_{CON}$	Vector of the exponents of the dynamic power consumed by the convolutional processors	Dimensionless	$3 \leq \gamma_{CON}(m) \leq 3.2$
$\vec{\gamma}_{CLA}$	Vector of the exponents of the dynamic power consumed by the classifier processors	Dimensionless	$3 \leq \gamma_{CLA}(m) \leq 3.2$
$\vec{P}_{CON}^{(IDLE)}$	Vector of the idle power consumed by the convolutional processors	(Watt)	$P_{CON}^{(IDLE)}(m) = 10^{-7}$
$\vec{P}_{CLA}^{(IDLE)}$	Vector of the idle power consumed by the classifier processors	(Watt)	$P_{CLA}^{(IDLE)}(m) = 10^{-7}$
$\vec{P}_{NET}^{(IDLE;Rx)}$	Vector of the idle power consumed by each receive port	(Watt)	$10^{-8} \leq P_{NET}^{(IDLE;Rx)}(m) \leq 10^{-7}$
$\vec{P}_{NET}^{(IDLE;Tx)}$	Vector of the idle power consumed by each transmit port	(Watt)	$10^{-8} \leq P_{NET}^{(IDLE;Tx)}(m) \leq 10^{-7}$
$\vec{\psi}$	Vector of the dimensionless Transport-to-Physical protocol overheads at m-th tier	Dimensionless	$\psi(m) = 1.105$
$\vec{\Omega}_{NET}^{(Rx)}$	Vector of the scaling coefficients of the dynamic power consumed by each transmit port	(Watt/(bit/s)$^{\zeta_{NET}^{(Tx)}}$)	$0 \leq \Omega_{NET}^{RX}(m) \leq 10^{-14}$
$\vec{\zeta}_{NET}^{(Rx)}$	Vector of the exponents of the dynamic power consumed by each receive port	Dimensionless	$2.1 \leq \zeta_{NET}^{(Rx)}(m) \leq 2.3$
$\vec{\Omega}_{NET}^{(Tx)}$	Vector of the scaling coefficients of the dynamic power consumed by each transmit port	(Watt/(bit/s)$^{\zeta_{NET}^{(Rx)}}$)	$0 \leq \Omega_{NET}^{RX}(m) \leq 10^{-14}$
$\vec{\zeta}_{NET}^{(Tx)}$	Vector of the exponents of the dynamic power consumed by each transmit port	Dimensionless	$2.4 \leq \zeta_{NET}^{(Rx)}(m) \leq 2.5$
$\vec{f}^{(MAX)}$	Vector of the maximum processing frequencies of the convolutional processors	(Mbit/s)	$f^{(MAX)}(m) = 9$
$\vec{\widetilde{f}}^{(MAX)}$	Vector of the maximum processing frequencies of the classifier processors	(Mbit/s)	$\widetilde{f}^{(MAX)}(m) = 8$
I_{MAX}	The maximum number of primal-dual iterations	Dimensionless	$I_{MAX} = 450$
a_{MAX}	Clipping factor of the step-sizes of the implemented primal-dual iterations	Dimensionless	$a_{MAX} = 1.3 \times 10^{-5}$
$iter\_number$	Total number of primal-dual iterations performed by each run of *DynDeFog_TRACKER*	Dimensionless	$iter\_number = 450$
$\vec{a}_{FogT}^{(MAX)}$	Vector of the clipping factors tested by each run of *DynDeFog_TRACKER*	Dimensionless	$10^{-6} \leq a_{FogT}^{(MAX)}(m) \leq 7 \times 10^{-6}$
$\overrightarrow{jump_1}$	Vector of the up/down multiplicative scaling factors applied to the scalar components of the input vectors of *DynDeFog_TRACKER*	Dimensionless	$0 \leq jump_1(m) \leq 1$
$\overrightarrow{jump_2}$	Vector of the up/down multiplicative scaling factors applied to the scalar components of the input vectors of *DynDeFog_TRACKER*	Dimensionless	$0 \leq jump_2(m) \leq 1$

3.2 THE OUTPUT PARAMETERS TO THE DEEPFOGSIM SIMULATOR

DeepFogSim uses a set of 22 global variables to store the output results along with some dummy intermediate outputs. Following Table B lists these output parameters of the *DeepFogSim* simulator, together with their meaning, and measuring units.

TABLE B. Output parameters of the *DeepFogSim* simulator version 4.0.

INPUT PARAMETERS	MEANING/ROLE	MEASURING UNITS
$[f_{mtx}]$	Matrix storing the time traces of the convolutional processing frequencies	(Mbit/s)
$\left[\widetilde{f}_{mtx}\right]$	Matrix storing the time traces of the classifier processing frequencies	(Mbit/s)
$[R_{mtx}]$	Matrix storing the time traces of the per-tier flows	(Mbit/s)
$[\lambda_{mtx}]$	Matrix storing the time traces of the Lagrange multipliers	(Joule)
$[\mathcal{E}_{TOT}]$	Matrix of the total energy	(Joule)
$[\mathcal{E}_{COP}]$	Matrix of the computation energy	(Joule)
$[\mathcal{E}_{NET}]$	Matrix of the network energy	(Joule)
$\left[\mathcal{E}_{TOT}^{(best)}\right]$	Matrix of the best total energy returned by *DynDeF_RAS*	(Joule)
$\left[\mathcal{E}_{COP}^{(best)}\right]$	Matrix of the best computation energy returned by *DynDeF_RAS*	(Joule)
$\left[\mathcal{E}_{NET}^{(best)}\right]$	Matrix of the best network energy returned by *DynDeF_RAS*	(Joule)
$\vec{f}^{(best)}$	Vector storing the best frequency values of the convolutional processors returned by *DynDeF_RAS*	(Mbit/s)
$\vec{\widetilde{f}}^{(best)}$	Vector storing the best frequency values of the classifier processors returned by *DynDeF_RAS*	(Mbit/s)
$\vec{R}^{(best)}$	Vector storing the best transport rates of the connections returned by *DynDeF_RAS*	(Mbit/s)
$\overrightarrow{perexit}_{delay}^{(ratios)}$	Vector of the per-exit actual-vs-maximum tolerated delay ratios returned by *DynDeF_RAS*	Dimensionless
$\left[\mathcal{E}_{TOT}^{(MAX)}\right]$	Matrix of the total energy obtained by the *Static_Allocation* function	(Joule)
$\left[\mathcal{E}_{COP}^{(MAX)}\right]$	Matrix of the computation energy obtained by the *Static_Allocation* function	(Joule)
$\left[\mathcal{E}_{NET}^{(MAX)}\right]$	Matrix of the network energy obtained by the *Static_Allocation* function	(Joule)
$\left[\mathcal{E}_{TOT}^{(Track)}\right]$	Matrix of the total energy returned by *DynDeFog_TRACKER*	(Joule)
$\left[\mathcal{E}_{COP}^{(Track)}\right]$	Matrix of the computation energy returned by *DynDeFog_TRACKER*	(Joule)
$\left[\mathcal{E}_{NET}^{(Track)}\right]$	Matrix of the network energy returned by *DynDeFog_TRACKER*	(Joule)
$\left[\lambda_{1}^{(Track)}\right]$	Matrix of the three traces of the first multiplier returned by *DynDeFog_TRACKER*	(Joule)
$\left[\lambda_{M}^{(Track)}\right]$	Matrix of the three traces of the last multiplier returned by *DynDeFog_TRACKER*	(Joule)

3.3 THE *DynDeF_RAS* FUNCTION

The *DynDeF_RAS* function:

```
DynDeF_RAS(f_0, f_tilde_0, R_0, lambda_0)
```

implements the primal-dual adaptive iterations in (25) and (27) for the numerical evaluation of the solution of the optimization problem in (6). The goal is to perform the optimized constrained allocation of the computing and networking resources needed by the simulated hierarchical Fog platform of Fig. 2 for sustaining the inference phase of the CDNN of Fig. 1a running atop it. The input parameters of this function are the following four vectors: $\left\{\vec{f}_0, \vec{\widetilde{f}}_0, \vec{R}_0, \vec{\lambda}_0\right\}$, which are needed for the initialization

of the primal-dual iterations in (24) and (25). The values of these vectors are set by calling the main script of the *DeepFogSim* simulator.

The *DynDeF_RAS* function returns, by saving the values to the corresponding global variables, the following output variables:

1. $[f_{mtx}] \in \mathbb{R}^{Q \times I_{MAX}}$ (bit/s): the matrix collecting the frequencies of the convolutional processors on a per-iteration basis;
2. $[\widetilde{f}_{mtx}] \in \mathbb{R}^{Q \times I_{MAX}}$ (bit/s): the matrix collecting the frequencies of the classifier processors on a per-iteration basis;
3. $[\boldsymbol{R}_{mtx}] \in \mathbb{R}^{(M-1) \times I_{MAX}}$ (bit/s): the matrix collecting the frequencies of the network throughputs on a per-iteration basis;
4. $[\lambda_{mtx}] \in \mathbb{R}^{M \times I_{MAX}}$ (Joule): the matrix collecting the *lambda* multipliers on a per-iteration basis;
5. $\overrightarrow{\mathcal{E}_{TOT}} \in \mathbb{R}^{1 \times I_{MAX}}$ (Joule): the vector collecting the total energy in (3) on a per-iteration basis;
6. $\overrightarrow{\mathcal{E}_{COP}} \in \mathbb{R}^{1 \times I_{MAX}}$ (Joule): the vector collecting the computing energy in (4) on a per-iteration basis; and,
7. $\overrightarrow{\mathcal{E}_{NET}} \in \mathbb{R}^{1 \times I_{MAX}}$ (Joule): the vector collecting the network energy in (5) on a per-iteration basis,

where the I_{MAX} parameter fixes the maximum number of the allowed primal-dual iterations.

The last column of the previous matrices and vectors are the optimized (best) values of the allocated resources and consumed energy, respectively: $\vec{f}^{(best)}$, $\vec{\widetilde{f}}^{(best)}$, $\vec{R}^{(best)}$, $\vec{\lambda}^{(best)}$, $\mathcal{E}_{TOT}^{(best)}$, $\mathcal{E}_{COP}^{(best)}$ and $\mathcal{E}_{NET}^{(best)}$, which represent the final output of the *DynDeF_RAS* function.

Algorithm 1 presents a pseudo-code of the *DynDeF_RAS* function. An examination of this code points out that the asymptotic computational complexity of the *DynDeF_RAS* function scales linearly with I_{MAX}.

3.4 THE *Static_Allocation* FUNCTION

The *Static_Allocation* function:

```
Static_Allocation()
```

calculates the computing, networking, and total energy consumed by the simulated Fog platform of Fig. 2 for sustaining the delay-constrained inference phase of the considered CDNN under the (static) maximal allocation vectors: $\left\{ \vec{f}^{(MAX)}, \vec{\widetilde{f}}^{(MAX)}, \vec{R}^{(MAX)} \right\}$.

Hence, the *Static_Allocation* function returns:

1. $\mathcal{E}_{TOT}^{(MAX)}$: the maximum total energy consumed by the the simulated Fog platform of Fig. 2 under the maximal resource vectors;
2. $\mathcal{E}_{COP}^{(MAX)}$: the maximum computing energy consumed by the the simulated Fog platform of Fig. 2 under the maximal resource vectors; and,
3. $\mathcal{E}_{NET}^{(MAX)}$: the maximum network energy consumed by the the simulated Fog platform of Fig. 2 under the maximal resource vectors.

Algorithm 2 presents a pseudo-code of the implemented *Static_Allocation* function. An examination of this code points out that the asymptotic computational complexity of the *Static_Allocation* function scales linearly with the sum of M of tiers and the total number Q of Fog nodes: i.e., $M + Q$.

3.5 THE *DynDeFog_TRACKER* FUNCTION

The numerical outputs of the *DynDeFog_TRACKER* function:

```
DynDeFog_TRACKER(f_0, f_tilde_0, R_0, lambda_0),
```

are three matrices that store the time-behaviors of:

1. the consumed total energy $E\_TOT(Joule)$;
2. the corresponding consumed network energy $E\_NET(Joule)$; and
3. the $lambda$ multiplier $(Joule)$,

Algorithm 1 — *DynDeF_RAS* function

function: $DynDeF\_RAS\left(\vec{f}_0,\ \vec{\tilde{f}}_0,\ \vec{R}_0,\ \vec{\lambda}_0\right)$.

Input: The initialized values: $\vec{f}_0,\ \vec{\tilde{f}}_0,\ \vec{R}_0,\ \vec{\lambda}_0$.

Output: $\vec{f}^{(best)}$, $\vec{\tilde{f}}^{(best)}$, and $\vec{R}^{(best)}$ vectors of the best resource allocation; scalar total energy $\mathcal{E}_{TOT}^{(MAX)}$ (Joule), scalar computing energy $\mathcal{E}_{COP}^{(MAX)}$ (Joule), and network energy $\mathcal{E}_{NET}^{(MAX)}$ (Joule) consumed under the optimized allocation vectors.

▷ **Begin DynDeF_RAS function**

for $m = 1 : M$ **do**
  for $j = 1 : m_m$ **do**
    Compute $v_I(j,m)$ with Eq. (1)
    Compute $v_O(j,m)$ with Eq. (2)
  end for
end for
for $t = 1 : I_{MAX}$ **do** ▷ **RAS iterations**
  for $m = 1 : M$ **do**
    for $j = 1 : m_m$ **do**
      Compute the $T_{EXE}(j,m)$ with Eqs. (7) and (8)
    end for
    Compute $T_{EXE}(m)$ with Eq. (9)
  end for
  Compute $T_{EXE}^{(1,m)}$ with Eq. (10)
  Compute $\mathcal{E}_{COP}$ with Eqs. (13), (15) and (4)
  Compute $\mathcal{E}_{NET}$ with Eq. (5)
  Compute $\mathcal{E}_{TOT}$ with Eq. (3)
  Compute all the derivatives of the Lagrangian function
  Update $\vec{f}$, $\vec{\tilde{f}}$, and $\vec{R}$ with Eqs. (24) and (25)
  Update $\vec{\lambda}$, $\vec{\alpha}$, and $\vec{\xi}$ with Eqs. (26) and (27)
end for
Obtain $\vec{f}^{(best)}$, $\vec{\tilde{f}}^{(best)}$, and $\vec{R}^{(best)}$
Compute $\mathcal{E}_{COP}^{(best)}$ with Eqs. (13), (15) and (4)
Compute $\mathcal{E}_{NET}^{(MAX)}$ with Eq. (5)
Compute $\mathcal{E}_{TOT}^{(MAX)}$ with Eq. (3)
return $\left[\vec{f}^{(best)},\ \vec{\tilde{f}}^{(best)},\ \vec{R}^{(best)},\ \mathcal{E}_{TOT}^{(best)},\ \mathcal{E}_{COP}^{(best)},\ \mathcal{E}_{NET}^{(best)}\right]$. ▷ **End DynDeF_RAS function**

for values of the iteration index going from: 1 to the given: *iteration_number*.

The goal of the *DynDeFog_TRACKER* function is to test the convergence rate to the steady state and the steady-state stability of the primal-dual iterations performed by the *DynDeF_RAS* function when unpredicted and abrupt changes in the operating conditions of the simulated Fog platform of Fig. 2 happen. These changes are formally dictated by the scaling vectors: $\overrightarrow{jump}_1$ and $\overrightarrow{jump}_2$ (see Table A), that multiply the scalar components of the input maximal resource allocation vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$.

Specifically, at the time indexes:

1. round $((1/5) \times iter\_number) + 1$;
2. round $((2/5) \times iter\_number) + 1$;

Algorithm 2 — *Static_Allocation* function

function: $Static\_Allocation()$.

Input: $FanIn(j,m)$, $FanOut(j,m)$, $\vec{f}^{(MAX)}$, $\vec{\tilde{f}}^{(MAX)}$, and $\vec{R}^{(MAX)}$.

Output: scalar total energy $\mathcal{E}_{TOT}^{(MAX)}$ (Joule), scalar computing energy $\mathcal{E}_{COP}^{(MAX)}$ (Joule), and network energy $\mathcal{E}_{NET}^{(MAX)}$ (Joule) consumed under the maximal allocation vectors.

▷ **Begin Static_Allocation function**

for $m = 1 : M$ **do**
  for $j = 1 : m_m$ **do**
    Compute $v_I(j,m)$ with Eq. (1)
    Compute $v_O(j,m)$ with Eq. (2)
    Compute the $T_{EXE}(j,m)$ with Eqs. (7) and (8)
  end for
  Compute $T_{EXE}(m)$ with Eq. (9)
end for
Compute $T_{EXE}^{(1,m)}$ with Eq. (10)
Compute $\mathcal{E}_{COP}^{(MAX)}$ with Eqs. (13), (15) and (4)
Compute $\mathcal{E}_{NET}^{(MAX)}$ with Eq. (5)
Compute $\mathcal{E}_{TOT}^{(MAX)}$ with Eq. (3)
return $\left[\mathcal{E}_{TOT}^{(MAX)}, \mathcal{E}_{COP}^{(MAX)}, \mathcal{E}_{NET}^{(MAX)}\right]$.

▷ **End Static_Allocation function**

3. round $((3/5) \times iter\_number) + 1$; and
4. round $((4/5) \times iter\_number) + 1$,

the initial values of the sliced resource vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$, undergo changes. These changes are obtained by multiplying (on a per scalar entry-basis) the components of the sliced vector $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$ by: $\overrightarrow{jump}_1$ and $\overrightarrow{jump}_2$.

The resulting time-traces of the computing energy, network energy, total energy, λ_1 multiplier and λ_M multiplier over the time interval: $[1, iter\_number]$ are stored (on a per-row basis) into the global matrix variables: $[\mathcal{E}_{COP}]$, $[\mathcal{E}_{NET}]$, $[\mathcal{E}_{TOT}]$, $[\lambda_1]$ and $[\lambda_M]$ for three values of the speed-up factor: a_{MAX} stored into the (1×3) input vector $\vec{a}_{MAX}$.

The feasibility of the operating conditions induced by $\overrightarrow{jump}_1$ and $\overrightarrow{jump}_2$ are explicitly tested by the *DynDeFog_TRACKER* function, and suitable terminating error messages are generated if infeasible operating conditions occur.

The input parameters of the *DynDeFog_TRACKER* function are the same ones of the *DynDeF_RAS* function, namely $\vec{f}_0$, $\vec{\tilde{f}}_0$, $\vec{R}_0$, and $\vec{\lambda}_0$.

The output of the *DynDeFog_TRACKER* function are returned into the global $(3 \times iter\_number)$ matrix variables: $\left[\lambda_1^{(Track)}\right]$, $\left[\lambda_M^{(Track)}\right]$, $\left[\mathcal{E}_{TOT}^{(Track)}\right]$, $\left[\mathcal{E}_{COP}^{(Track)}\right]$ and $\left[\mathcal{E}_{NET}^{(Track)}\right]$.

Specifically, the *DynDeFog_TRACKER* function proceeds as follows:

1. it checks the feasibility of the operating conditions induced by $\overrightarrow{jump}_1$ and $\overrightarrow{jump}_2$. If both the operating conditions are feasible, then,
2. it begins to run the *DynDeF_RAS* function over the time interval: $[1, iter\_number/5]$ under the original setting of $\vec{v}_0$, $\vec{f}^{(MAX)}$, $\vec{\tilde{f}}^{(MAX)}$ and $\vec{R}^{(MAX)}$;
3. at the end of iteration $iter\_number/5$, *all* the scalar entries of the sliced vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$ are element-wise multiplied by $\overrightarrow{jump}_1$;
4. the *DynDeF_RAS* function runs over the interval: $[(iter\_number/5) + 1, (2/5) \times iter\_number]$ under the setting of step 3);

5. at the end of iteration $(2/5)\times iter\_number$, the entries of the sliced vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$ are restored to their original values by dividing them by $\overrightarrow{jump}_1$;
6. the *DynDeF_RAS* function runs over the interval: $[(2/5)\times iter\_number+1, (3/5)\times iter\_number]$ under the setting of step 5);
7. after iteration $(3/5)\times iter\_number$, *all* the scalar entries of the sliced vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$ are element-wise multiplied by $\overrightarrow{jump}_2$;
8. the *DynDeF_RAS* function runs over the interval: $[(3/5)\times iter\_number+1, (4/5)\times iter\_number]$ under the setting of step 7);
9. at the end of iteration $(4/5)\times iter\_number$, the entries of the sliced vector: $\left[\vec{v}_0, \vec{f}^{(MAX)}, \vec{\tilde{f}}^{(MAX)}, \vec{R}^{(MAX)}\right]$ are restored to their original values by dividing them by $\overrightarrow{jump}_2$; and, finally,
10. the *DynDeF_RAS* function runs over the final interval: $[(4/5)\times iter\_number+1, iter\_number]$ under the setting of step 9).

Graphic plots of the time traces of the returned matrices $\left[\lambda_1^{(Track)}\right]$, $\left[\lambda_M^{(Track)}\right]$, $\left[\mathcal{E}_{TOT}^{(Track)}\right]$, $\left[\mathcal{E}_{COP}^{(Track)}\right]$ and $\left[\mathcal{E}_{NET}^{(Track)}\right]$ are displayed at the end of the *DynDeFog_TRACKER* run. From the previous description, it follows that the asymptotic complexity of the *DynDeFog_TRACKER* function scales up as in: $\mathcal{O}(8\times iteration\_number\times 3)$. Graphic plots of the time-behaviors of the returned E_TOT, E_NET, and the first and last $lambda$ multipliers are displayed at the end of each $FogTracker$ run (see Section 5). From the outset, it follows that the asymptotic complexity of the *FogTracker* function scales up as in: $\mathcal{O}((2(Q+M)-1)\times iteration\_number\times 3)$.

3.6 AUXILIARY FUNCTIONS OF *DEEPFOGSIM*

The main script of the actual version 4.0 of *DeepFogSim* and the previous three main routines call some auxiliary functions. In this subsection, we provide a detailed description of these auxiliary functions.

3.6.1 THE Check_convexity AUXILIARY FUNCTION

The *Check_convexity* function

```
Check_convexity()
```

checks for the (strict) *convexity* of the optimization problem in (6). If the problem is not strictly convex, the function returns an error message and then exits the program.

3.6.2 THE Check_feasibility AUXILIARY FUNCTION

The *Check_feasibility* function

```
Check_feasibility()
```

tests the (strict) *feasibility* of the underlying constrained optimization problem in (6) by checking M delay-induced constraints in (6b). If at least one of these M constraints fails, the function generates an error message indicating the failed delay constraint and then stops the overall program.

3.6.3 THE Check_input AUXILIARY FUNCTION

The *Check_input* function

```
Check_input()
```

checks for the formal validity of the input data set by the user. If any input variable is not formally valid, the function returns an error message and then exits the program.

3.6.4 THE init_other_global AUXILIARY FUNCTION

The *init_other_global* function

```
init_other_global()
```

initializes all the output and dummy (global) variables that have not been directly set by the user in the configuration script.

3.6.5 THE oneDtotwoD AUXILIARY FUNCTION

The *oneDtotwoD* function allows to transform the mono-dimensional (i.e., string type) representation of the network topology of Fig. 2 to the corresponding bi-dimensional one. It is used to map a sequential scalar number falling in the range $1, \dots, Q$ into the bi-dimensional index of the i-th Fog node lying in the j-th tier.

Specifically, the function:

```
[tier, column] = oneDtotwoD(oneDindex)
```

converts the 1D indexing of a Fog node to the corresponding 2D row/column one. The input parameter $oneDindex$ must be integer and falling in the range: $1, \dots, Q$. The returned $tier$ parameter is integer and falls in the range: $1, \dots, M$. The returned corresponding $column$ parameter is integer and falls in the range: $1, \dots, m_m$

3.6.6 THE twoDtooneD AUXILIARY FUNCTION

The *twoDtooneD* function allows to transform the bi-dimensional representation of the network topology of Fig. 2 to the corresponding one-dimensional (i.e., string type) one. It is used to map the bi-dimensional index of the i-th Fog node lying in the j-th tier into a sequential scalar number falling in the range $1, \dots, Q$.

Specifically, the function:

```
oneDindex = twoDtooneD(tier, column)
```

evaluates the 1D equivalent index of the 2D indexing of a Fog node. The returned $oneDindex$ is integer valued and fall in the range: $1, \dots, Q$. The value of the tier parameter must be integer and falling in the range: $1, \dots, M$. The value of the column parameter must be integer and falling in the interval: $1, \dots, m_m$.

3.7 COMPUTATIONAL COMPLEXITY OF *DEEPFOGSIM*

Table C presents a synoptic view of the asymptotic computational complexities of the described *DynDeF_RAS*, *Static_Allocation* and *DynDeFog_TRACKER* functions.

TABLE C. Asymptotic computational complexities of the main functions implemented by the version 4.0 of parallel *DeepFogsim*.

Function	***Asymptotic computational complexity***
DynDeF_RAS	$\mathcal{O}\left(I_{MAX}\,(2(Q+M)-1)\right)$
Static_Allocation	$\mathcal{O}\,(2(Q+M)-1)$
DynDeFog_TRACKER	$\mathcal{O}\,((2(Q+M)-1)\times iter\_number\times 3)$

4 THE SUPPORTED DUAL-MODE USER INTERFACES

The current version of the simulator supports two user interfaces, referred to as *DeepFogSim* and *DeepFogSim Graphic User Interface (DeepFogSimGUI)*. As detailed in the following two sub-sections, both interfaces make available the same set of basic optimization routines of Table C, so that they provide the same set of numerical results. However, we remark that:

1. the *DeepFogSim* interface is oriented to a scientific use of the simulator. Its utilization requires some basics about the MATLAB environment. Hence, it may be appealing for skilled research users, who desire to work in an interactive way and are mainly interested to check and optimize the performance of the Fog execution platform of Fig. 2;

2. the *DeepFogSimGUI* interface provides a rich set of self-explicative ready-to-use native facilities that allow less (or even not) skilled users to directly run the simulator under a number of pre-loaded (but, in any case, customizable) application scenarios. Hence, since its utilization does not require any specific skill about the MATLAB environment, it allows the user to interact with the simulator as a "black-box". For this purpose, (i) the obtained numerical results are rendered in form of colored plots, in order to make more intuitive their interpretation and mining; and, (ii) an easy-to-consult online version of the User Guide is also enclosed.

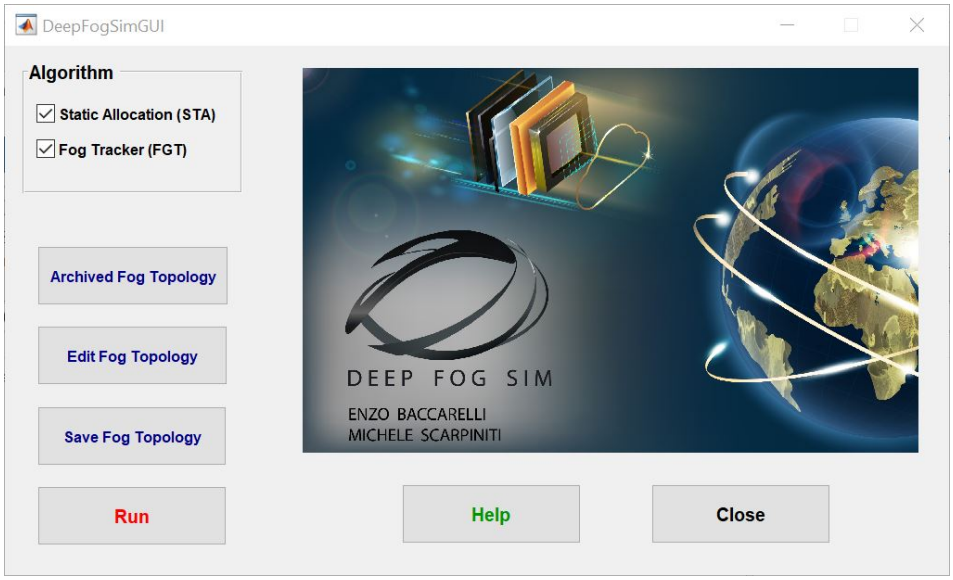


FIGURE 5. A screenshot of the GUI interface.

4.1 THE *DeepFogSim* INTERFACE

This interface is activated by entering the command:

```
DynDeepFogSim
```

in the command line of a running MATLAB session. The script acts as the main program of the *DeepFogSim* package. This script allows the user to select any subset of the natively supported optimization algorithms by setting some binary flags to 1 (for running it) or to 0 (for not running it), and to choose which Fog topology should be run by writing the name of the related configuration script. If the user desires to change the topology configuration, the script defining the Fog topology can be opened in the editor window of MATLAB.

By programming in the MATLAB language, it allows the user to:

1. define the desired simulation setup by editing the input parameters listed in Table A;
2. select any subset of the supported optimization functions of Table C;
3. use the online *help* of MATLAB, in order to display information about any specific function supported by the *DeepFogSim* package; and,
4. write and call user-defined customized functions that are not natively supported by the current version of the simulator.

Depending of the selected functions (see Table C), after running *DeepFogSim*, the obtained numerical results are displayed in tabular form, and/or as colored time plots, and/or as colored bar plots (see Section 5 for additional details about the graphic rendering capabilities and related options supported by the simulator).

4.2 THE *DeepFogSimGUI* INTERFACE

The GUI interface is opened by entering the command:

```
DeepFogSimGUI
```

in the command line of a running MATLAB session. The screenshot of the displayed graphic window is shown in Fig. 5.

This interface is supported by the MATLAB code: *DynDeepFogSim_Run*, that acts as the main script of the overall simulator.

An examination of Fig. 5 points out that the GUI interface of the simulator supports seven pre-built functions (namely, *Help*, *Algorithm*, *Archived Fog Topology*, *Edit Fog Topology*, *Save Fog Topology*, *Run* and *Close*), that can be activated by the user by clicking over the corresponding bottoms. Table D lists these native GUI functions and points out their meaning and associated actions.

TABLE D. Native functions supported by the version 4.0 of the *DeepFogSimGUI* interface and associated actions.

Available GUI Functions	***Associated Actions***
Help	Allow to access the User Guide of the simulator by opening a dedicated PDF file.
Algorithm	Allow to select any subset of the natively supported optimization algorithms by clicking the corresponding labels.
Archived Fog Topology	Allow to retrieve an already archived Fog topology with the corresponding simulation setup, in order to run it.
Edit Fog Topology	Allow to edit a new Fog topology and/or a new simulation setup by compiling the list of input parameters of Table A.
Save Fog Topology	Allow to save the lastly edit Fog topology and assign it an identification label.
Run	Allow to run the selected optimization algorithm under the retrieved/edit Fog topology and associated simulation setup.
Close	Shut down the current working session of the *DeepFogSim* simulator and close all the figure windows.

From the user perspective, a typical *DeepFogSimGUI*-enabled working session should proceed according to the following ordered steps:

1. type `DeepFogGUI` at the MATLAB prompt, in order to open the GUI interface;
2. click the ***Help*** bottom, in order to access the PDF version of the User Guide. This step is optional;
3. select any desired subset of the supported optimization procedures by clicking the corresponding labels in the ***Algorithm*** window. This step is mandatory;
4. retrieve an already stored Fog Topology in *.mat format and the associated list of input parameters by accessing the ***Archived Fog Topology***. Alternatively, build up a new set of the desired input parameters of Table A by clicking the ***Edit Fog Topology*** bottom. This step is mandatory;
5. store in the ***Archived Fog Topology*** database the lastly edit setup in *.mat format by clicking the ***Save Fog Topology*** bottom. An identification label for the ***Save Fog Topology*** is required. This step is optional;
6. click the ***Run*** bottom, in order to start the execution of the selected algorithms under the (retrieved or edit) selected setup. At the end of the execution, the attained results are returned in tabular form, and/or as colored time plots, and/or as colored bar plots (see Section 5 for additional details about the graphic rendering capabilities and related options supported by the simulator);
7. after finishing the current working session of the simulator, click the ***Close*** bottom, in order to shut down the *DeepFogSimGUI* interface.

5 THE SUPPORTED GRAPHIC FORMATS OF THE RENDERED DATA

Under the current version 4.0 of the simulator, both the *DeepFogSim* and *DeepFogSimGUI* interfaces support four main formats, in order to render the results output by the three optimization algorithms of Table C. These rendering formats are:

- the **Tabular** format. It is enabled by the *print_solution* graphic function;
- the **Colored Graphic Plot** format. It is enabled by the *plot_solution* graphic function;
- the **Colored Time-tracker Plot** format. It is enabled by the *plot_FogTracker* graphic function;
- the **Fog Topology** format. It is enabled by the *plot_Topology* graphic function.

Table E gives a synoptic view of the rendering formats available for each optimization algorithm, together with the required supporting rendering functions.

TABLE E. Formats of the rendered data done available by the version 4.0 of the *DeepFogSim* and *DeepFogSimGUI* interfaces and supporting rendering functions.

	Tabular Format	***Colored Graphic Plot***	***Colored Time-tracker Plot***	***Topology Graph Plot***	***Supporting Rendering Functions***
DynDeF_RAS	✓	✓	✗	✓	• *print_solution* • *plot_solution* • *plot_Topology*
Static_Allocation	✓	✓	✗	✓	• *print_solution* • *plot_solution* • *plot_Topology*
DynDeFog_TRACKER	✗	✗	✓	✗	• *plot_FogTracker*

As illustrative examples, the following Figs. 6, 7 and 8 report the screenshots of the Tabular, Colored Graphic and Topology Graph rendering formats we have obtained by running the *DynDeF_RAS* algorithm under a test topology. In particular,

i. Fig. 6 reports the numerical values of all involved optimization variables of Eq. (6), together with the corresponding values of the consumed total: $\mathcal{E}_{TOT}$, computing: $\mathcal{E}_{COP}$ and network: $\mathcal{E}_{NET}$ energies;

ii. Fig. 7 reports the corresponding: (i) task allocation (upper part); (ii) computing frequencies and consumed bandwidths (middle part); and, (iii) consumed energies (bottom part), in form of colored bar plots; and,

iii. Fig. 8 reports a labeled version of the considered DAG, in which the colors of the DAG nodes correspond to the allocation of the tasks actually performed by *DynDeF_RAS* on the Mobile (Green color), Fog (Red color) and Cloud (Azure color) computing infrastructures.

Similar screenshots are rendered when the *Static_Allocation* algorithm is run (see the second row of Table E).

Finally, Fig. 9 is an example of the Colored Time-tracker plots that are obtained by running the *DynDeFog_TRACKER* function in correspondence of three values of the step-size a_{MAX} (see Table A). Specifically, the four plots of Fig. 9 report the time behaviors of $\mathcal{E}_{TOT}$, $\mathcal{E}_{NET}$, λ_1, and λ_M, i.e., the first and last multiplier. These plots are obtained under the (aforementioned) illustrative scenario.

In the following sub-sections, we pass to (shortly) describe the syntax and semantic of the rendering functions listed in the last column of Table E.

5.1 THE *print_solution* rendering function

The *print_solution* function:

```
print_solution(strategy)
```

prints on the MATLAB prompt the result obtained by running the tested strategy under the selected topology and given input parameters (see Table A).

This function has an optional input parameter that allows the print of the results obtained by the *DynDeF_RAS* function and the *Static_Allocation* function, respectively.

The function prints the following results in a numerical form:

1. the optimized frequencies $\{f_{jm}\}$ of the convolutional processors, for $j = 1, \ldots, m_m$ and $m = 1, \ldots, M$;
2. the optimized frequencies $\{\tilde{f}_{jm}\}$ of the classifier processors, for $j = 1, \ldots, m_m$ and $m = 1, \ldots, M$;

```
Resource allocation performed by DynDeFRAS

Best convolutional frequencies:
f(1, 1):  4473889.088 (Mb/s)
f(2, 1):  4473889.088 (Mb/s)
f(3, 1):  4473889.088 (Mb/s)
f(4, 1):  4473889.088 (Mb/s)
f(1, 2):  6968203.186 (Mb/s)
f(2, 2):  6968203.186 (Mb/s)
f(1, 3):  7547804.498 (Mb/s)

Best classifier frequencies:
f(1, 1):  4685886.495 (Mb/s)
f(2, 1):  4685886.495 (Mb/s)
f(3, 1):  4685886.495 (Mb/s)
f(4, 1):  4685886.495 (Mb/s)
f(1, 2):  5971957.566 (Mb/s)
f(2, 2):  5971957.566 (Mb/s)
f(1, 3):  6296604.751 (Mb/s)
----------------------------
E_TOT_best:      369.641 (J)
E_COP_best:      333.477 (J)
E_NET_best:       36.164 (J)
E_NET / E_TOT:     9.78  (%)

End run of DynDeF_RAS

Start run of Static_Allocation
Network with 3 tiers and 7 Fog nodes
Resource allocation performed by Static_Allocation:
E_TOT_MAX:     1960.928 (J)
E_COP_MAX:     1882.157 (J)
E_NET_MAX:       78.770 (J)
E_NET_MAX / E_TOT_MAX: 4.02 (%)

E_TOT_best / E_TOT_MAX: 18.85 (%)

End run of Static_Allocation

Start run of the Discontinuous version of DynDeFog_TRACKER
Loop: 1/3
Loop: 2/3
Loop: 3/3
End run of the Discontinuous version of DynDeFog_TRACKER
```

FIGURE 6. An illustrative screenshot of the rendered Tabular format.

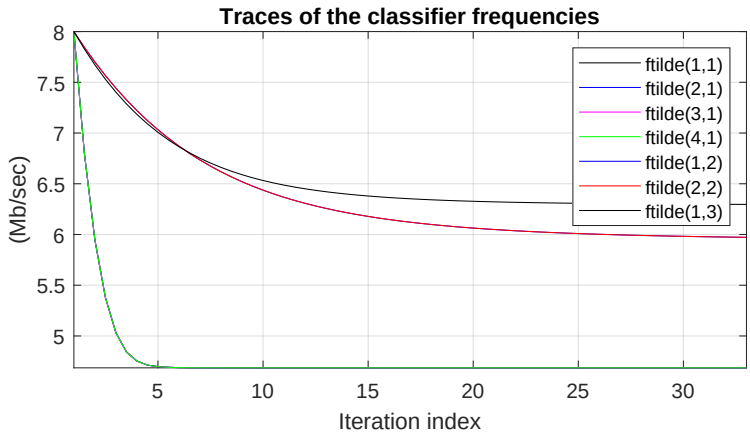
3. the optimized energies $\mathcal{E}_{TOT}^{(best)}$, $\mathcal{E}_{COP}^{(best)}$ and $\mathcal{E}_{NET}^{(best)}$;
4. the maximum energies $\mathcal{E}_{TOT}^{(MAX)}$, $\mathcal{E}_{COP}^{(MAX)}$ and $\mathcal{E}_{NET}^{(MAX)}$ consumed by the *Static_Allocation*, if it is selected this option in calling the main script;
5. the ratio $\mathcal{E}_{NET}^{(best)}/\mathcal{E}_{TOT}^{(best)}$ and/or the ratio $\mathcal{E}_{NET}^{(MAX)}/\mathcal{E}_{TOT}^{(MAX)}$, depending on which option is selected.

The *print_solution* function prints on the MATLAB prompt also the total computing time needed for running all the selected strategies.

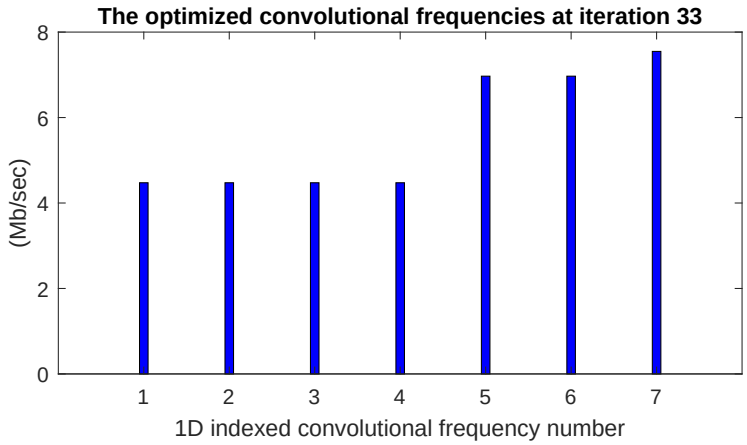
5.2 THE *plot_solution* rendering function

The *plot_solution* function:

```
fignumber = plot_solution(fignumber, options)
```



(a)



(b)

FIGURE 7. Two illustrative screenshots of the rendered Colored Graphic Plot format.

opens a number of figure windows, which graphically display the numerical results obtained by the *DeepFogSim* simulator.

This function accepts two optional input parameters, i.e.: (i) the number of figure from which to start; and, (ii) a flag used to choose if the results returned by the *Static_Allocation* function must be also plotted. A maximum of eleven different figures may be opened.

Specifically, this function displays:

1. a figure collecting the traces of the Q convolutional frequencies $\{f_{jm}\}$;
2. a figure collecting the traces of the Q classifier frequencies $\{\tilde{f}_{jm}\}$;
3. a figure collecting the traces of the $M-1$ throughputs $\{R_m\}$;
4. a figure collecting the traces of the *lambda* multipliers $\{\lambda_m\}$;
5. a figure collecting in three subplots the traces of $\mathcal{E}_{TOT}^{(best)}$, $\mathcal{E}_{COP}^{(best)}$ and $\mathcal{E}_{NET}^{(best)}$;
6. a figure showing a bar plot of the Q optimized convolutional frequencies $\{f_{jm}^{(best)}\}$;
7. a figure showing a bar plot of the optimized classifier frequencies $\{\tilde{f}_{jm}^{(best)}\}$;
8. a figure showing a bar plot of the $M-1$ optimized throughputs $\{R_m^{(best)}\}$;
9. a figure showing a bar plot of the $\mathcal{E}_{TOT}^{(best)}$, $\mathcal{E}_{COP}^{(best)}$ and $\mathcal{E}_{NET}^{(best)}$;

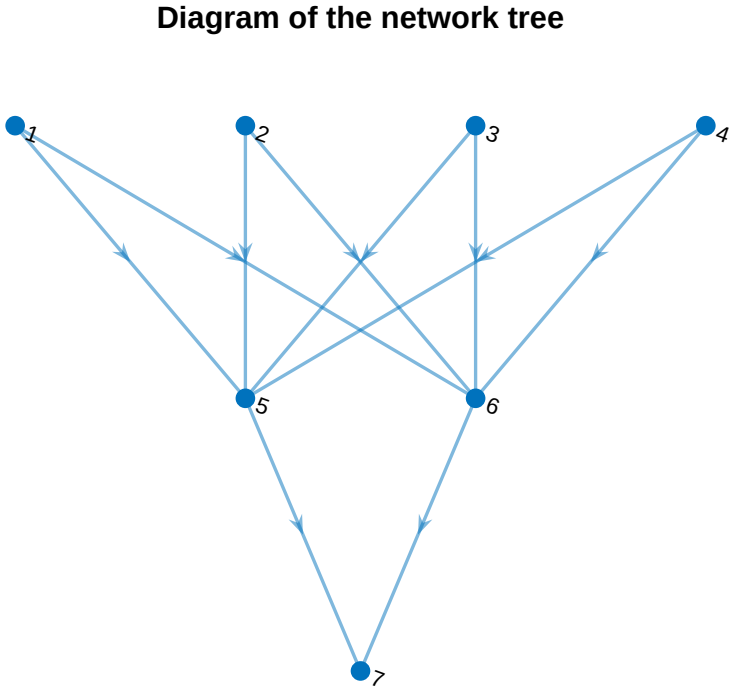


FIGURE 8. An illustrative screenshot of the rendered Topology Graph format.

10. a figure showing a bar plot of the per-exit actual-vs.-maximum tolerated delay ratios; and
11. (optionally) a figure showing a bar plot of the $\mathcal{E}_{TOT}^{(MAX)}$, $\mathcal{E}_{COP}^{(MAX)}$ and $\mathcal{E}_{NET}^{(MAX)}$ consumed by the *Static Allocation* strategy.

5.3 THE *plot_Topology* rendering function

The *plot_Topology* function:

```
plot_Topology()
```

returns a graphic representation of the network topology set in the input configuration for the Fog platform to be simulated. This function reads the global matrix variable $[A]$, which represents the *adjacency matrix* of the considered topology.

5.4 THE *plot_FogTracker* rendering function

The *plot_FogTracker* function:

```
fignumber = plot_FogTracker(fignumber)
```

provides the graphic capabilities needed for a proper plot of the time-traces of the total energy, computing energy and networking energy, and the first and last lambda multipliers generated by the *DynDeFog_TRACKER* function under the three values of step-size that are stored by the input vector $\vec{a}_{FogT}^{(MAX)}$ (see Table A). Its input parameter is the number of figure from which to start, to be sequential from the last plotted figure. Specifically, the function renders five figures that orderly report:

1. the total consumed computing-plus-networking energy $\left[\mathcal{E}_{TOT}^{(Track)}\right]$ under the three step-size values stored by the input $\vec{a}_{FogT}^{(MAX)}$ vector;
2. the corresponding consumed computing energy $\left[\mathcal{E}_{COP}^{(Track)}\right]$ under the three step-size values stored by the input $\vec{a}_{FogT}^{(MAX)}$ vector;
3. the corresponding consumed networking energy $\left[\mathcal{E}_{NET}^{(Track)}\right]$ under the three step-size values stored by the input $\vec{a}_{FogT}^{(MAX)}$ vector;

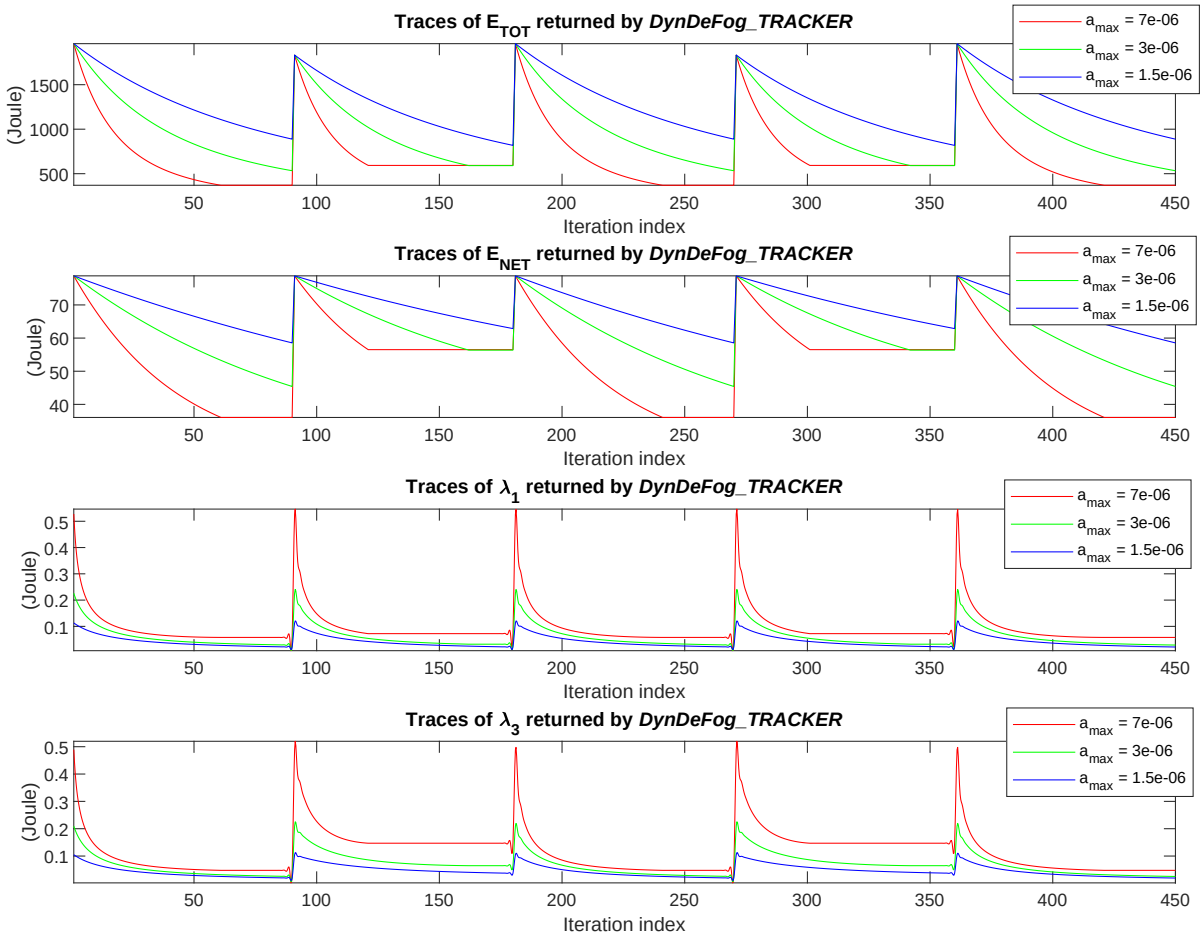


FIGURE 9. An illustrative screenshot of the time-tracker plots rendered by a run of the *DynDeFog_TRACKER* function.

4. the related values of the first lambda multiplier $\left[\lambda_1^{(Track)}\right]$ under the three step-size values stored by the input $\vec{a}_{FogT}^{(MAX)}$ vector; and,
5. the related values of the last lambda multiplier $\left[\lambda_M^{(Track)}\right]$ under the three step-size values stored by the input $\vec{a}_{FogT}^{(MAX)}$ vector.

6 THE SET OF PRE-LOADED APPLICATION TOPOLOGIES

In the current version 4.0 of the *DeepFogSimGUI* interface (see Fig. 5 in Section 4), it is available an archive that stores eight test setups, together with the related sets of (suitably tuned) input parameters. These network topologies are ready-for-the-use, i.e., they may be retrieved by the user and, then, run under both the (previously described) interfaces of the simulator.

The archived network setups cover several tree-based topologies, with different number of tiers and per-tier nodes. Specifically, the number of Fog nodes ranges from $Q = 7$, to $Q = 63$, the number of tiers ranges from $M = 3$ to $M = 6$, while the number of link varies in the range $[10, 682]$.

The remaining part of this section reports the adjacency matrix $[A]$ together with the graphs of the corresponding topology for all the saved setups. The first two topologies share the same tree-based network, hence we only show one of them.

6.1 Topology 2

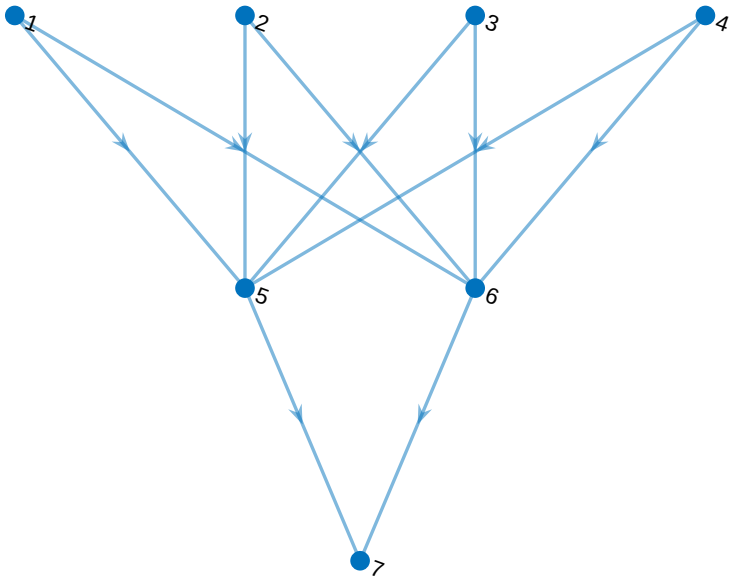


FIGURE 10. Topology 2

$$[A] = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

6.2 Topology 3

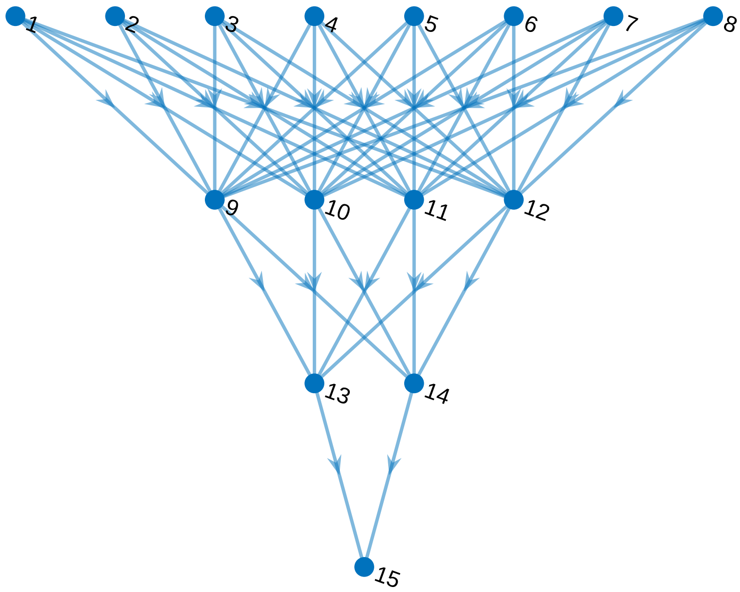


FIGURE 11. Topology 3

$$[A] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

6.3 Topology 4

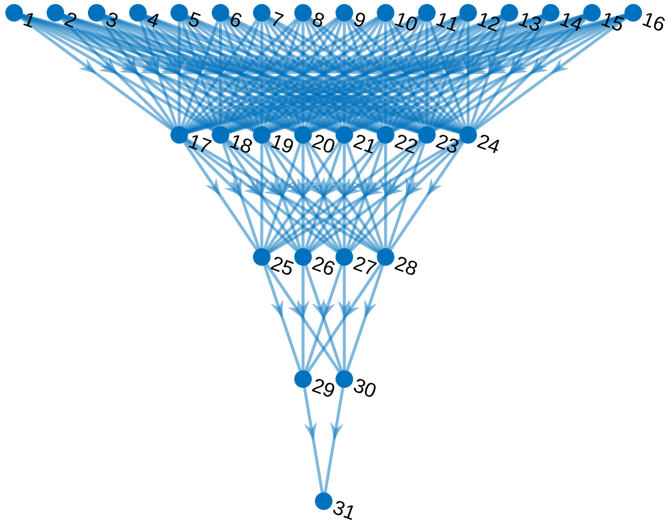


FIGURE 12. Topology 4

$$
[A] = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},
$$

6.4 Topology 5

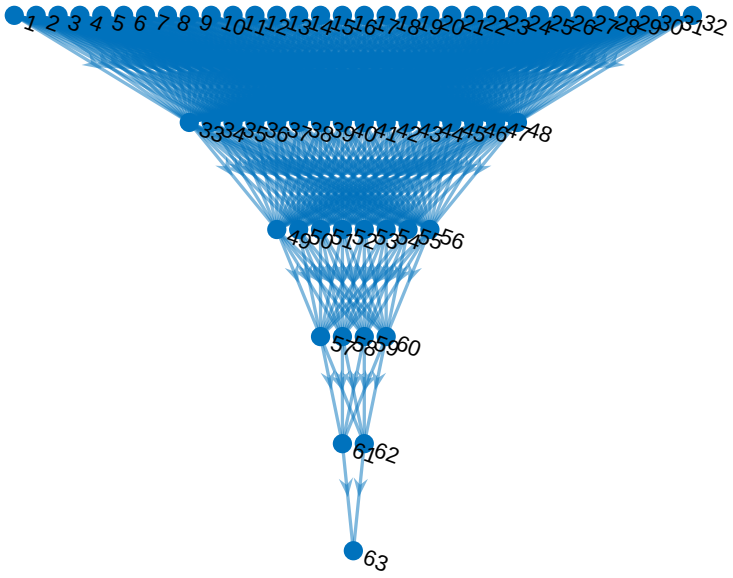


FIGURE 13. Topology 5

The adjacency matrix $[A]$ of Topology 5 can be written in the following general form:

$$[A] = \begin{bmatrix} \mathbf{0}_{32\times32} & \mathbf{1}_{32\times16} & \mathbf{0}_{32\times8} & \mathbf{0}_{32\times4} & \mathbf{0}_{32\times2} & \mathbf{0}_{32\times1} \\ \mathbf{0}_{16\times32} & \mathbf{0}_{16\times16} & \mathbf{1}_{16\times8} & \mathbf{0}_{16\times4} & \mathbf{0}_{16\times2} & \mathbf{0}_{16\times1} \\ \mathbf{0}_{8\times32} & \mathbf{0}_{8\times16} & \mathbf{0}_{8\times8} & \mathbf{1}_{8\times4} & \mathbf{0}_{8\times2} & \mathbf{0}_{8\times1} \\ \mathbf{0}_{4\times32} & \mathbf{0}_{4\times16} & \mathbf{0}_{4\times8} & \mathbf{0}_{4\times4} & \mathbf{1}_{4\times2} & \mathbf{0}_{4\times1} \\ \mathbf{0}_{2\times32} & \mathbf{0}_{2\times16} & \mathbf{0}_{2\times8} & \mathbf{0}_{2\times4} & \mathbf{0}_{2\times2} & \mathbf{1}_{2\times1} \\ \mathbf{0}_{1\times32} & \mathbf{0}_{1\times16} & \mathbf{0}_{1\times8} & \mathbf{0}_{1\times4} & \mathbf{0}_{1\times2} & \mathbf{0}_{1\times1} \end{bmatrix},$$

where $\mathbf{0}_{n\times m}$ and $\mathbf{1}_{n\times m}$ denote matrices of dimension $n \times m$ full of zeros and ones, respectively.

6.5 Topology 6

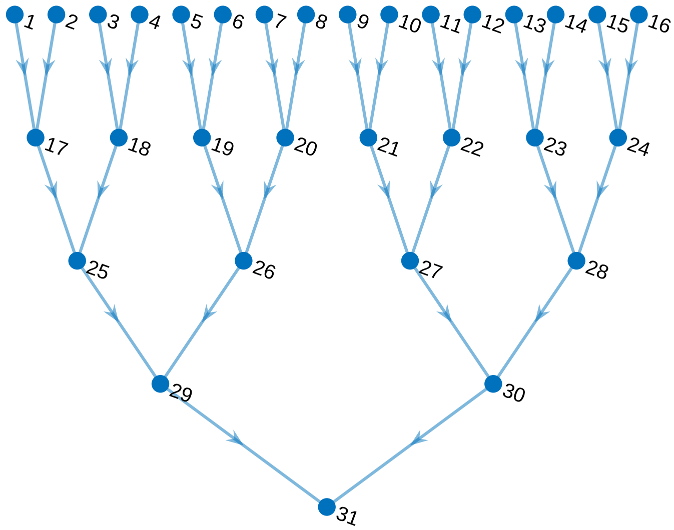


FIGURE 14. Topology 6

$$
[A] = \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix},
$$

6.6 Topology 7

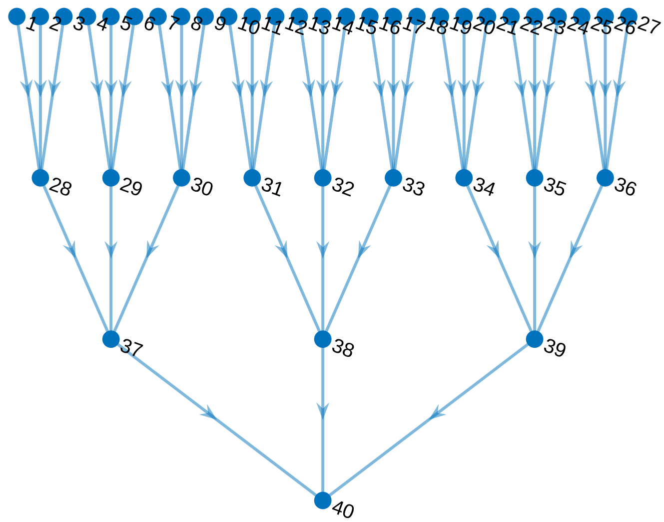


FIGURE 15. Topology 7

$$
[A] = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},
$$

6.7 Topology 8

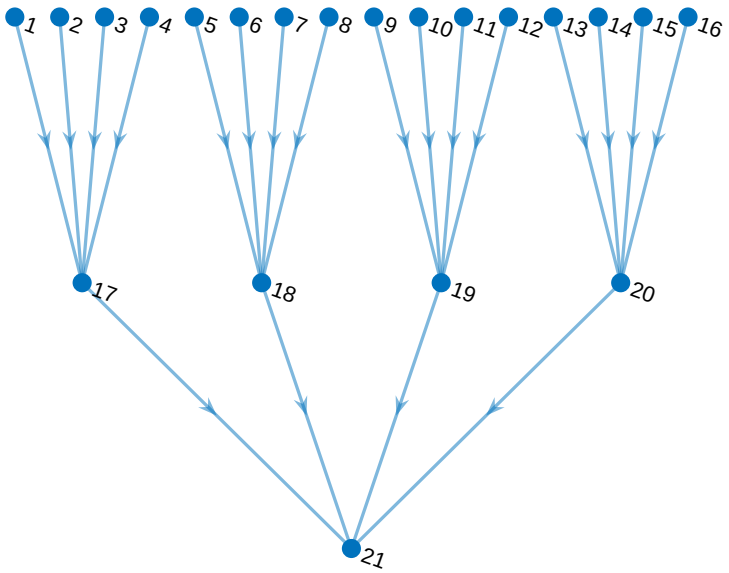


FIGURE 16. Topology 8

$$[A] = \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix},$$

7 AVAILABILITY OF THE DEEPFOGSIM SOFTWARE PACKAGE

The complete software package of the *DeepFogSim* simulator and the corresponding full *User Guide* may be downloaded *for free* at:

https://github.com/mscarpiniti/DeepFogSim

AKNOWLEDGEMENTS

This work has been partially supported by the projects: “SoFT: Fog of Social IoT” funded by Sapienza University of Rome Bando 2018 and 2019; “End-to-End Learning for 3D Acoustic Scene Analysis (ELeSA)” funded by Sapienza University of Rome

Bando Acquisizione di medie e grandi attrezzature scientifiche 2018; and, “DeepFog – Optimized distributed implementation of Deep Learning models over networked multitier Fog platforms for IoT stream applications” funded by Sapienza University of Rome Bando 2020.

References

[1] E. Baccarelli, P. G. Vinueza Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, “Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study,” *IEEE Access*, vol. 5, pp. 9882–9910, 2017.

[2] E. Baccarelli, S. Scardapane, M. Scarpiniti, A. Momenzadeh, and A. Uncini, “Optimized training and scalable implementation of conditional deep neural networks with early exits for Fog-supported IoT applications,” *Information Sciences*, vol. 521, pp. 107–143, June 2020.

[3] R. Priyadarshini, R. K. Barik, and H. Dubey, “DeepFog: Fog computing-based deep neural architecture for prediction of stress types, diabetes and hypertension attacks,” *Computation*, vol. 6, no. 4, p. 62, 2018.

[4] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, “Why should we add early exits to neural networks?,” *Cognitive Computation*, vol. 12, pp. 954–966, September 2020.

[5] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, and J. Henry, *IoT Fundamentals-Networking Technologies, Protocols, and Use Cases for the internet of Things*. Cisco Press, 2017.

[6] E. Baccarelli, M. Scarpiniti, and A. Momenzadeh, “EcoMobiFog – Design and dynamic optimization of a 5G Mobile-Fog-Cloud multi-tier ecosystem for the real-time distributed execution of stream applications,” *IEEE Access*, vol. 7, pp. 55565–55608, 2019.

[7] E. Baccarelli, M. Biagi, R. Bruno, M. Conti, and E. Gregori, “Broadband wireless access networks: A roadmap on emerging trends and standards,” in *Broadband Services: Business Models and Technologies for Community Networks*, ch. 14, pp. 215–240, Wiley Online Library, October 2005.

[8] E. Baccarelli and M. Biagi, “Power-allocation policy and optimized design of multiple-antenna systems with imperfect channel estimation,” *IEEE Transactions on Vehicular Technology*, vol. 53, pp. 136–145, January 2004.

[9] E. Baccarelli, M. Biagi, C. Pelizzoni, and N. Cordeschi, “Optimized power-allocation for multiantenna systems impaired by multiple access interference and imperfect channel estimation,” *IEEE Transactions on Vehicular Technology*, vol. 56, pp. 3089–3105, May 2007.

[10] Q. Peng, A. Walid, J. Hwang, and S. H. Low, “Multipath TCP: Analysis, design, and implementation,” *IEEE/ACM Transactions on Networking*, vol. 24, pp. 596–609, February 2016.

[11] M. Scarpiniti, E. Baccarelli, and A. Momenzadeh, “VirtFogSim: A parallel toolbox for dynamic energy-delay performance testing and optimization of 5G Mobile-Fog-Cloud virtualized platforms,” *Applied Sciences*, vol. 9, p. 1160, March 2019.