electronics

MDPI

Article

Intelligent Semantic Communication System Based on Kolmogorov–Arnold Networks Driven by Dynamic Terminal-Side Computing Power Network

Wang Liu [1], Qingtao Zeng [1,*], Likun Lu [1] and Waheed Abdul [2]

[1] School of Information Engineering, Beijing Institute of Graphic Communication, Beijing 102600, China; a256970@163.com (W.L.)
[2] School of International Education, Beijing Institute of Graphic Communication, Beijing 102600, China; laserspc_tech@yahoo.com
* Correspondence: zengqingtao@bigc.edu.cn

Abstract: With the advent of the 6G era, the number of IoT devices has experienced explosive growth, leading to the generation of massive amounts of data at the network edge. Semantic communication, as an innovative solution to handling this data deluge, can significantly enhance communication efficiency. However, the limited storage and computational resources of terminal devices constrain the widespread application of semantic communication in 6G networks. To address this issue, we propose a terminal-side-computing-driven intelligent semantic communication solution. Specifically, we introduce a semantic communication model based on Kolmogorov–Arnold Networks (KANs), named K-DeepSC. Using image-reconstruction tasks as an example, the proposed K-DeepSC reduces the number of model parameters by 44% compared to semantic communication models based on Multi-Layer Perceptrons (MLPs), while maintaining similar performance. Furthermore, to fully leverage idle terminal computing power for semantic tasks, we explore computation offloading in dynamic Terminal-Side Computing Power Networks. By optimizing task delay minimization, a deep reinforcement learning algorithm is employed to determine the optimal offloading strategy. Simulation results demonstrate that our proposed solution effectively reduces semantic task processing delay.

Keywords: terminal-side computing power network; semantic communication; reinforcement learning; task offloading

check for updates

Citation: Liu, W.; Zeng, Q.; Lu, L.; Abdul, W. Intelligent Semantic Communication System Based on Kolmogorov–Arnold Networks Driven by Dynamic Terminal-Side Computing Power Network. *Electronics* **2024**, *13*, 4076. https://doi.org/10.3390/electronics13204076

Academic Editor: Christos J. Bouras

Received: 25 September 2024
Revised: 14 October 2024
Accepted: 15 October 2024
Published: 17 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the rapid development of cutting-edge technologies such as artificial intelligence, 5G, and the Internet of Things (IoT), new applications like smart healthcare, intelligent manufacturing, and vehicle networking are continually emerging. These advancements not only change human lifestyles but also present new challenges and demands for traditional communication networks. According to IDC forecasts, by 2025, the number of IoT devices worldwide is expected to reach 41.6 billion, generating 79.4 zettabytes (ZB) of data. Efficiently processing and transmitting such massive amounts of data has become a critical issue that future communication networks must address.

In recent years, semantic communication [1] has shown great potential in reducing data transmission volumes and improving communication efficiency. Semantic communication systems based on deep learning, relying on high-performance deep neural networks (DNNs) such as Convolutional Neural Networks (CNNs) and Transformer architectures, have demonstrated outstanding system performance. However, because DNNs typically involve a large number of parameters and require high computational power, these systems have very high demands on storage and computational resources. This is particularly challenging for terminal devices, whose limited storage and computational capabilities significantly restrict the practical effectiveness of semantic communication. Therefore, it is urgent

to develop effective solutions to improve the feasibility of semantic communication systems on terminal devices. To address this critical problem, this paper proposes a solution that utilizes terminal-side computing networks to support semantic communication, thereby improving the performance of semantic communication systems in resource-constrained environments. The main contributions of this paper are as follows:

1. A solution enabling semantic communication through terminal-side computing power networks is proposed to address the challenges of limited storage and computational resources faced by terminal applications of semantic communication.
2. We introduce a semantic communication model based on Kolmogorov–Arnold Networks, integrating a novel linear attention mechanism and DropKey technology, named K-DeepSC, using image-reconstruction tasks in smart healthcare as an example. Compared to the MLPs-based DeepSC, K-DeepSC significantly reduces the number of model parameters while maintaining similar performance.
3. A multi-autonomous network architecture is proposed for computation offloading scenarios in dynamic Terminal-Side Computing Power Networks, with a primary terminal (PT) responsible for decision-making. By establishing a multi-constraint optimization model aimed at minimizing the processing delay of semantic computing tasks and employing the deep reinforcement learning algorithm DQN for a solution, experimental results demonstrate that this approach excels in reducing latency.

2. Related Work

Semantic communication is currently in its preliminary research phase. For example, in the literature [2], academic Zhang Ping proposed the Semantic Base (Seb) model and, based on this, constructed the Intelligent and Efficient Semantic Communication (IE-SC) model. He also pointed out the vast potential of semantic communication in edge scenarios such as intelligent manufacturing and smart unmanned aerial vehicle (UAV) networks. Subsequently, in the literature [3], the metric system for semantic information was further summarized, and two methods for semantic encoding transmission, direct encoding, and transform encoding were discussed. In the field of deep learning-enabled semantic communication research, reference [4] proposed a semantic communication system called DeepSC, based on the Transformer architecture, specifically for text transmission tasks. Compared to traditional communication systems, DeepSC demonstrates greater robustness against channel variations. Subsequently, a significant amount of research has been conducted to further explore and expand upon the DeepSC system [5–12]. To address the issue of limited resources on terminal devices, several related studies have been conducted. For instance, in the literature [13], Xie et al. proposed a lightweight distributed semantic communication system, L-DeepSC, for text transmission tasks. This model employs two model compression techniques—network sparsification (pruning weights) and network quantization (reducing weight resolution)—to reduce the size and complexity of the trained model, thereby lowering transmission and hardware costs for Internet of Things (IoT) devices. The literature [14] proposes a novel semantic communication paradigm—AMJSCC—specifically designed for image transmission tasks. By introducing a semantic adaptive model selection module, this paradigm enables the flexible selection of optimal models under varying conditions, including different models, channels, and transmission objectives, thereby effectively reducing storage space and communication resource consumption. In the literature [15], an H.264-like compression method is proposed to address the compression transmission of GNSS information in autonomous vehicles. This method leverages the principle of similarity between adjacent frames found in H.264 encoding and incorporates Huffman coding with error correction capabilities, thereby improving compression efficiency and effectively conserving communication resources in vehicular networks. In the literature [16], Zhang et al. proposed a unified semantic communication model, U-DeepSC, for multi-task scenarios. This model allows terminal devices to store only a single model capable of performing up to six different tasks, greatly enhancing system resource efficiency and flexibility.

The Computing Power Network (CPN) [17] aims to interconnect widely distributed computing resources through a network, achieving unified coordination and on-demand allocation of multidimensional resources such as networking, computing, and storage. As one of the key technologies for future 6G networks, the Computing Power Network has garnered extensive attention and research in recent years. For example, in [18], Tang et al., inspired by the concept of Intent-Based Networking (IBN), proposed a service intent-aware task-scheduling framework for the Computing Power Network and developed a bidding-based task scheduling algorithm to achieve optimal matching between task intents and computing resources. Zheng et al. [19] constructed dynamic resource pools within the Computing Power Network and employed an attention-based Proximal Policy Optimization (APPO) algorithm to evaluate the contribution of different resource pools to the system, thereby maximizing the utilization of idle computing resources. Zhou et al. [20] designed an intelligent computing power network architecture that deeply integrates CPN with autonomous networks, summarizing its three core features: intent-driven, closed-loop autonomy, and self-adaptive evolution.

With the rapid increase in the number of terminal devices, the role of terminal devices in computing networks is becoming increasingly important. To fully leverage the widely distributed computing power of these terminal devices, the literature [21] proposed an edge-extended computing network architecture—Terminal-Side Computing Power Network—and discussed its architecture and key technologies in detail. Additionally, the literature [22] proposed a semantic communication framework based on the CPN, aimed at utilizing ubiquitous computing resources in the network to support semantic communication systems, and demonstrated the advantages and potential application scenarios of this framework through specific use cases.

3. Terminal-Side Computing Network Enabled Semantic Communication System Model

3.1. Semantic Communication Model

3.1.1. Model Description

(1) Transmitter: The Transmitter consists of a semantic encoder module and a channel encoder module. We use a Transformer encoder to construct the semantic encoder. First, the image is converted into an embedding sequence Z_{in} through the embedding layer, where Z_{in} is the sequence of patches input to the semantic encoder, P is the number of patches, and D is the embedding dimension; its formula is as follows:

$$Z_{in} = [z_1, z_2, \ldots, z_N], \quad Z_{in} \in R^{P \times D} \tag{1}$$

To ensure the stability of deep neural network training, residual connections are employed between the modules in both the encoder and decoder. In Figure 1, these residual connections are represented by the $\oplus$ symbol. Each semantic encoder consists of a Focused Linear Attention layer and a Kolmogorov–Arnold Networks layer. KANs are a novel neural network architecture based on the Kolmogorov–Arnold representation theorem [23]. The Kolmogorov–Arnold representation theorem states that any continuous multivariate function $f(x_1, x_2, \ldots, x_n)$ can be represented as a finite sum of continuous single-variable functions, as shown in (2).

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{qp}(x_p) \right) \tag{2}$$

where Φ_q and $\psi_{qp}(x_p)$ are fixed continuous single-variable functions. The index p ranges from 1 to n, corresponding to the inputs x_1 to x_n, while q represents the lower bound of the number of univariate functions required to approximate any n-dimensional continuous function, which is $2n + 1$. Based on the Kolmogorov–Arnold representation theorem, unlike traditional MLPs, KANs place learnable activation functions at the edges of the network rather than at the nodes. These activation functions are typically parameterized as

spline functions, which endows the network with greater nonlinear expressiveness. This design replaces the traditional linear weight matrices, significantly reducing the number of network parameters while providing high flexibility. As a result, KANs can achieve the performance of MLPs with a large number of parameters using a smaller number of parameters when modeling complex nonlinear relationships in data.

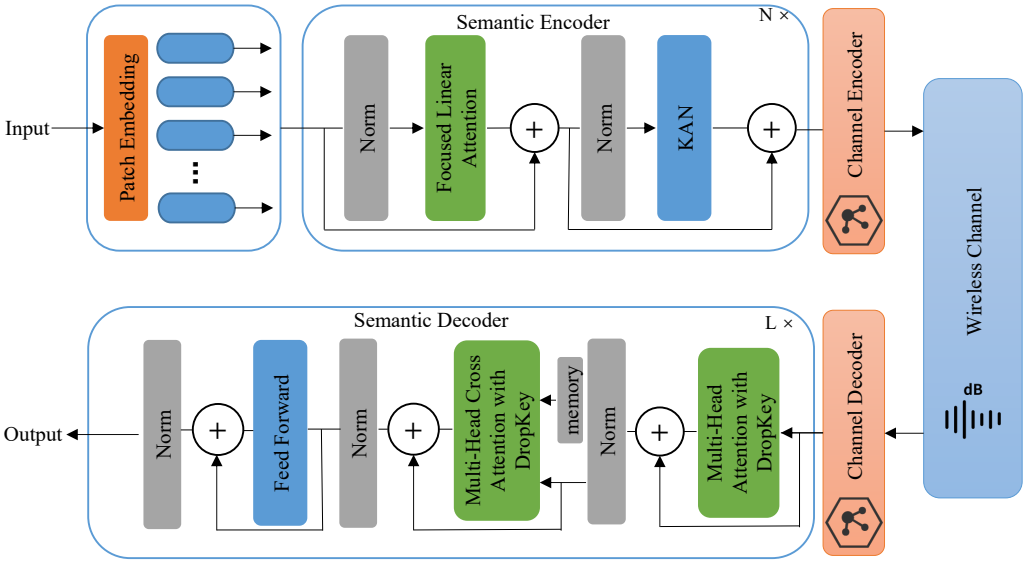


Figure 1. Proposed semantic communication model K-DeepSC.

Additionally, to reduce the computational complexity of the semantic encoding module, we employ a novel Focused Linear Attention (FLA) mechanism as a replacement for the conventional Softmax attention mechanism [24]. Specifically, this mechanism adjusts the directions of query and key features using a simple mapping function $f_p(x)$, which brings similar query-key pairs closer together, as shown in (3).

$$f_p(x) = \frac{\|x\|}{\|x^{**p}\|} \cdot x^{**p} \tag{3}$$

where x^{**p} represents taking the p-th power of each element of the vector x. Through the function $f_p(x)$, important features in the image can receive greater attention. Overall, the encoding process of a semantic encoder can be represented as

$$Z_{out} = Z_{in} + FLA(Z_{in}) + KAN(Z_{in} + FLA(Z_{in})) \tag{4}$$

In addition, in each layer of the semantic encoder, normalization is performed before the FLA and KAN modules, with the number of layers $N = 12$. Subsequently, the channel encoder performs channel encoding on Z_{out}, generating the transmitted signal S_{in}.

(2) Wireless Channel: The signal S_{in} is transmitted through a noisy wireless channel, and the receiver receives the signal S_{out}, as shown in Equations (5) and (6).

$$S_{out} = S_{in} + noise \tag{5}$$

$$S_{out} = ((S_{in} \cdot H) + noise) \cdot H^{-1} \tag{6}$$

Equation (5) represents the Additive white Gaussian noise (AWGN) channel, and Equation (6) represents the Rayleigh fading channel. Here, *noise* refers to Gaussian white noise with a mean of 0 and variance σ^2, and $\mathrm{H} = \begin{pmatrix} H_{\text{real}} & -H_{\text{imag}} \\ H_{\text{imag}} & H_{\text{real}} \end{pmatrix}$ represents the Rayleigh fading channel matrix, where H_{real} and H_{imag} represent the real and imaginary parts of the

channel matrix H, respectively. Their values are generated from a normal distribution with a mean of 0 and a variance of $\frac{1}{2}$.

(3) Receiver: The received signal S_{out} is first decoded by the channel decoding module to produce the signal $\hat{Z}_{in}$, which is then passed to the semantic decoding module. The semantic decoding module follows the Transformer architecture, primarily consisting of multi-head attention mechanisms and feedforward networks. To mitigate overfitting in the semantic communication model, we have improved the multi-head attention mechanism by introducing a novel regularizer, DropKey [25]. The decoding process is shown in Equation (7). For clarity, the residual connections are omitted in the formula.

$$\hat{Z}_{out} = FFN(MCAD(MSAD(\hat{Z}_{in}), memory)) \tag{7}$$

where *memory* contains contextual information about the input sequence. Additionally, normalization is applied after both the attention module and the feedforward network (FFN) module, with the number of layers $L = 8$.

3.1.2. Loss Function and Performance Metric

For image-reconstruction tasks, we use the mean squared error (MSE) as the loss function for the semantic communication model, as shown in Equation (8).

$$\text{MSELoss} = \frac{1}{l}\sum_{i=1}^{l}(y_i - \widehat{y_i})^2 \tag{8}$$

where l represents the sample length. The Peak Signal-to-Noise Ratio (PSNR) is used to evaluate the quality of the transmitted image, as shown in Equation (9).

$$\text{PSNR} = 20 \cdot \log_{10}\left(\frac{\text{PIXEL_MAX}^2}{\text{MSE}}\right) \tag{9}$$

where PIXEL_MAX $= 255$ is the maximum pixel value of the image, and MSE is the mean squared error between the two images.

3.2. Terminal-Side Computing Power Network System Model

3.2.1. Task Model

When local devices are unable to independently complete semantic computing tasks, tasks can be offloaded to other computing terminals in order to better utilize the available computational resources. We envision a dynamic terminal-side computing power network scenario, which includes an edge server and three autonomous computing power networks (ACPNs). In each ACPN, the primary terminal (PT) is responsible for collecting information about the computational resources and task requirements of all devices within its own network. Additionally, the PT can obtain information about the computational resources of other ACPNs from the edge server, allowing it to make centralized task-offloading decisions.

In this scenario, the computing devices are classified into three categories: edge servers, PTs, and secondary terminals (STs). As shown in Figure 2, the terminals transmit task-related raw data and processed semantic information through wireless links. It is important to note that STs cannot communicate directly with each other. Computing terminals can either be fixed devices (such as computational routers) or mobile devices such as personal smart devices (with low mobility) or vehicles (with high mobility). During each time slot τ, the PT senses the status of its ACPN and generates a semantic task. It then decides which terminal to offload the task to, based on the computational resources available in both its own ACPN and other ACPNs.

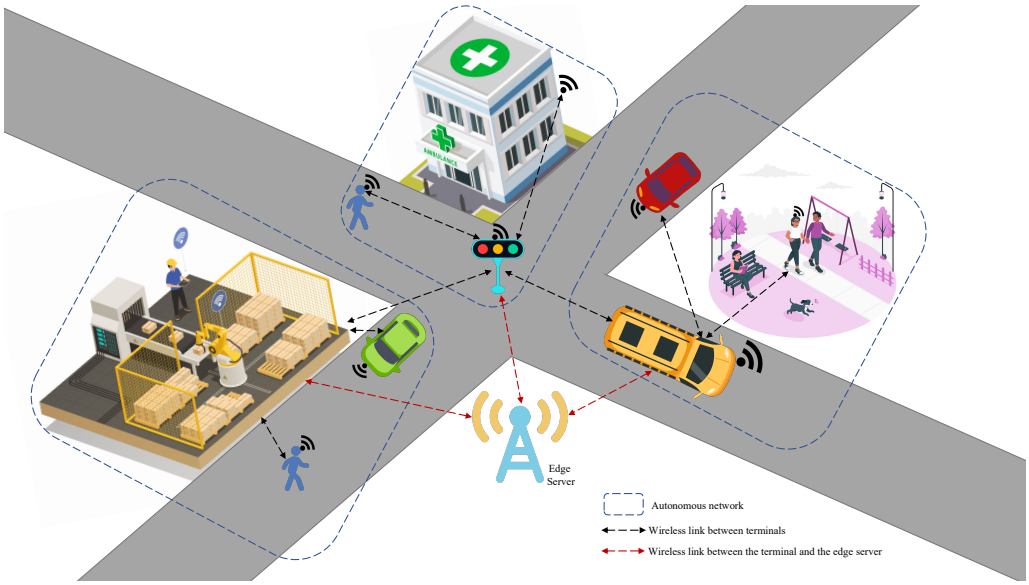


Figure 2. Dynamic terminal-side computing power network system model.

3.2.2. Computation Model

(1) Compute Tasks on Primary Terminal: Considering that frequent changes in the location of PTs in the autonomous network can lead to additional power consumption and thus reduce task completion rates, we fix the positions of PT in the autonomous network. Additionally, PTs can communicate directly with each other or through multi-hop indirect communication. The total delay for offloading to a PT includes computation delay, data communication delay, queuing delay, and result transmission delay. The computation delay is denoted by $t^C_{p,p}$ in Equation (10).

$$t^C_{p,p} = \frac{c_t}{p_c} \tag{10}$$

where c_t represents the computational load of the task, and p_c represents the computing capacity of the PT. The data communication delay is denoted by $t^T_{p,p}$ in Equation (11), and is given by

$$t^T_{p,p} = \begin{cases} 0 & \text{if local} \\ \beta \cdot I_{p,p} & \text{otherwise} \end{cases} \tag{11}$$

where β represents the number of hops and $I_{p,p}$ represents the communication delay between two adjacent PTs.

The queuing delay consists of two parts: the computation delay of tasks that are already in the queue of the current terminal and the delay caused by competition among multiple new tasks arriving at the terminal. This is denoted by $t^W_{p,p}$ in Equation (12). Here, $\mathcal{L}$ represents the tasks that are unfinished in the current terminal's queue, and $\mathcal{M}$ represents the tasks submitted to the current terminal in the current time slot.

$$t^W_{p,p} = \sum_{i=1}^{\mathcal{L}} \frac{c_i}{p_c} + \sum_{j=1}^{\mathcal{M}-1} \frac{c_j}{p_c} \tag{12}$$

Considering that the data after semantic encoding are much smaller than the original data, the delay required for result transmission is negligible. Therefore, the total delay for offloading to a PT is as shown in Equation (13).

$$t^{p2p}_{p,p} = t^C_{p,p} + t^T_{p,p} + t^W_{p,p} \tag{13}$$

(2) Compute Tasks on Secondary Terminal: Task offloading to STs can be divided into three stages: selecting the offloading terminal, determining the target ACPN, and

dispatching the task. Since PTs cannot communicate directly with STs in other autonomous networks, it is necessary to first check whether the target terminal is within the current ACPN after determining the offloading target. If it is not within the current network, the task must first be forwarded to the PT of the target ACPN, which will then be dispatched to the target terminal. Otherwise, the task is directly offloaded to the target terminal. Similar to Equation (13), the total processing delay includes computation delay, queuing delay, and data transmission delay, while the result transmission delay is negligible. The data communication delay is denoted by $t^C_{p,s}$ in Equation (14), where s_t is the task size and $\bar{tr}_{p2s}$ is the average transmission rate between the PT and the target ST.

$$t^T_{p,s} = \begin{cases} \frac{s_t}{\bar{tr}_{p2s}} & \text{if in the same autonomous network} \\ \frac{s_t}{\bar{tr}_{p2s}} + \beta I_{p,p} & \text{otherwise} \end{cases} \tag{14}$$

$$t_{conn} = \frac{\sqrt{r_p^2 - (y_p - y_s)^2} \pm (x_p - x_s)}{|v|} \tag{15}$$

$$R(t) = B \cdot \log_2 \left(1 + \frac{TP_p \cdot \delta}{NP \cdot \sqrt{(x_p - (x_s + v \cdot t))^2 + (y_p - y_s)^2}} \right) \tag{16}$$

$$\bar{tr}_{p2s} = \frac{1}{t_{conn}} \int_0^{t_{conn}} R(t)\, dt \tag{17}$$

Due to the mobility of terminal devices, it is necessary to consider the communication link time and communication rate between the PT and the target ST. Communication is only possible during the link connection time; otherwise, it is interrupted. Let t_{conn} be the communication link maintenance time between the target ST and its corresponding PT, where r_p represents the coverage radius of the PT, v denotes the speed of the target ST, and (x_p, y_p) and (x_s, y_s) are the coordinates of the PT and the target ST, respectively. The symbols $\pm$ indicate whether the target terminal is approaching or moving away from the PT. $R(t)$ represents the instantaneous communication rate at time t, where B is the communication bandwidth between the PT and the ST, TP_p is the transmission power of the PT, δ is the path loss factor, and NP is the noise power. Similar to (10) and (12), the computation delay and waiting delay are denoted by $t^C_{p,s}$ and $t^W_{p,s}$ in (18) and (19), respectively, where s_c represents the computational capability of the ST.

$$t^C_{p,s} = \frac{c_t}{s_c} \tag{18}$$

$$t^W_{p,s} = \sum_i^{\mathcal{L}} \frac{c_i}{s_c} + \sum_j^{\mathcal{M}-1} \frac{c_j}{s_c} \tag{19}$$

Thus, the total delay from the PT to the ST can be expressed by Equation (20).

$$t^{p2s}_{p,s} = t^C_{p,s} + t^T_{p,s} + t^W_{p,s} \tag{20}$$

(3) Compute Tasks on Edge Server: Considering the vast computational resources and powerful computing capability of the edge server, the queuing delay and computation delay during the server's processing are therefore neglected. Since the amount of data after semantic encoding is relatively small, the transmission delay is also negligible. Consequently, the total delay when offloading to the edge server is composed solely of data communication delay, represented by $t^{p2e}_{p,e}$ in Equation (21), where tr_{p2e} denotes the transmission rate between the PT and the edge server.

$$t^{p2e}_{p,e} = \frac{s_t}{tr_{p2e}} \tag{21}$$

3.2.3. Problem Formulation

In each time slot τ, the total processing delay for all tasks generated by the PT is given by Equation (22), where P and S denote the number of PTs and STs, respectively. The binary decision variable $u_{p,n}$ represents whether the primary terminal p offloads a task to terminal n. We transform this offloading problem into a multi-constraint optimization problem with the goal of maximizing the task completion rate while minimizing the task execution delay. Therefore, the problem is modeled as shown in Equation (24).

$$T_p = \sum_{n=1}^{P+S+1} u_{p,n} t_{p,n}^{p2n}, \tag{22}$$

where $t_{p,n}^{p2n}$ is defined as follows:

$$t_{p,n}^{p2n} = \begin{cases} t_{p,p}^{p2p}, & \text{if } 1 < n \leq P, \\ t_{p,s}^{p2s}, & \text{if } P < n \leq P+S, \\ t_{p,e}^{p2e}, & \text{if } n = P+S+1. \end{cases} \tag{23}$$

Then, the preceding problem is formulated:

$$\begin{aligned} &\min \sum_{p=1}^{P} T_p \\ &\text{s.t.} \\ &\text{C1: } T_p \leq t_{conn}, \quad p \in \mathcal{P} \\ &\text{C2: } T_p \leq t_{tc}, \quad p \in \mathcal{P} \\ &\text{C3: } \overline{tr_{p2s}} \geq t_{rmin}, \quad p \in \mathcal{P},\ s \in \mathcal{S} \\ &\text{C4: } T_p > 0, \quad p \in \mathcal{P} \\ &\text{C5: } u_{p,n} = \begin{cases} 1, & \text{if the task is assigned by } p \text{ to } n, \\ 0, & \text{otherwise} \end{cases}, \quad p \in \mathcal{P} \\ &\text{C6: } \sum_{n=1}^{P+S+1} u_{p,n} = 1, \quad p \in \mathcal{P} \end{aligned} \tag{24}$$

Using the sets $\mathcal{P} = \{1, 2, \ldots, P\}$ and $\mathcal{S} = \{1, 2, \ldots, S\}$ to represent the primary and secondary terminals, respectively. Constraint C1 indicates that the task delay must be within the communication link connection time, while constraint C2 specifies that the task must be completed within the stipulated time t_{tc}. Constraint C3 ensures that the task transmission rate between primary and secondary terminals must exceed the minimum task transmission rate tr_{min}; otherwise, data transmission will fail. Constraints C4 to C6 ensure that the task is successfully offloaded to a terminal.

4. DQN-Based Terminal-Side Semantic Task Computation Offloading Algorithm

Considering the dynamics of ACPNs, the problem in (24) is described as a Markov Decision Process (MDP) and solved using deep reinforcement learning methods. The MDP can be represented as a triplet $(\mathbb{S}, \mathbb{A}, \mathbb{R})$, where $\mathbb{S}$ is the state space of the environment, $\mathbb{A}$ is the action space, and $\mathbb{R}$ is the reward function.

4.1. MDP-Based Problem

State Space: The state space contains the information observed from the environment. The state information S_τ is represented in (25), where $S_\tau \in \mathbb{S}$, and c_i represents the total computation workload of a terminal waiting for task processing observed by the agent at time slot τ.

$$S_\tau = [c_1, c_2, \ldots, c_{P+S+1}] \tag{25}$$

Action Space: The action space consists of the offloading decisions generated by all PTs in the time slot τ, denoted by A_τ in Equation (26), where $A_\tau \in \mathbb{A}$.

$$A_\tau = \left[a_1^1, \ldots, a_{P+S+1}^1, \ldots, a_1^2, \ldots, a_{P+S+1}^2, \ldots, a_1^p, \ldots, a_{P+S+1}^p\right] \tag{26}$$

Reward Function: In the time slot τ, after the agent interacts with the environment, it obtains the state S_τ. In this state, the agent chooses the action A_τ, which alters the environment state, resulting in the state changing to $S_{\tau+1}$ at time $\tau+1$. In the new state, the environment provides feedback reward R_τ to the agent. The agent then makes a decision based on R_τ and $S_{\tau+1}$ to choose a new action $A_{\tau+1}$. We define the negative value of the total processing delay of tasks in time slot τ as the reward, aiming to maximize this reward. Additionally, if the constraints of problem (24) are not met, the agent incurs a fixed penalty.

4.2. DQN-Based Computation Offloading Algorithm

The DQN-based terminal-side semantic task offloading algorithm architecture is shown in Figure 3. Q-function represents the long-term reward obtained by taking a specific action in a particular state. DQN uses a Deep Neural Network (DNN) to approximate the Q-function. The DNN accepts the state as input and outputs the Q-values for each possible action. The agent consists of a policy network and a target network, both implemented as a multi-layer perceptron. The policy network is responsible for selecting the optimal action based on the current state, while the target network serves as a copy of the policy network, with its parameters periodically updated to match those of the policy network to enhance training stability. In time slot τ, the agent interacts with the environment to obtain the current state S_τ. S_τ is fed into the policy network to obtain the predicted Q-values for all actions, and the action A_τ with the highest predicted Q-value is selected. The environment executes the action A_τ and provides a reward R_τ, while updating the state to $S_{\tau+1}$. A state transition sequence $\{S_\tau, A_\tau, R_\tau, S_{\tau+1}\}$ is generated and stored in the experience replay buffer. The agent randomly samples a mini-batch of transition sequences from the buffer and inputs them into the policy network and target network to obtain the predicted Q-values and target Q-values. The MSE is used as the loss function to update the parameters of the policy network. After a fixed number of episodes, the parameters of the target network are updated with the real-time parameters of the policy network. The target Q-value y_τ is calculated as shown in Equation (27).

$$y_\tau = R_\tau + \gamma \cdot \max_{a \in \mathbb{A}} Q(S_{\tau+1}, a) \tag{27}$$

where γ is the discount factor. The pseudo-code of our proposal based on DRL method DQN is presented in Algorithm 1. The total time complexity of Algorithm 1 is shown in expression (28).

$$O\left(E_p \cdot \left(N_{ac} + M_{batch} \cdot K_{params}\right)\right) \tag{28}$$

where E_p is the number of training episodes. Within a single episode, the time complexity for Step 4, which senses the environment, is $O(1)$. The time complexity for Step 5, which selects an action, is $O(N_{ac})$, where N_{ac} is the size of the action space. The time complexity for Steps 6–7 is $O(1)$. The complexity of updating the network in Step 8 is determined by the batch size M_{batch} and the number of network parameters K_{params}, resulting in a complexity of $O(M_{batch} \cdot K_{params})$. The complexity of parameter synchronization in Step 9 is related to the number of network parameters, with a complexity of $O(K_{params})$.

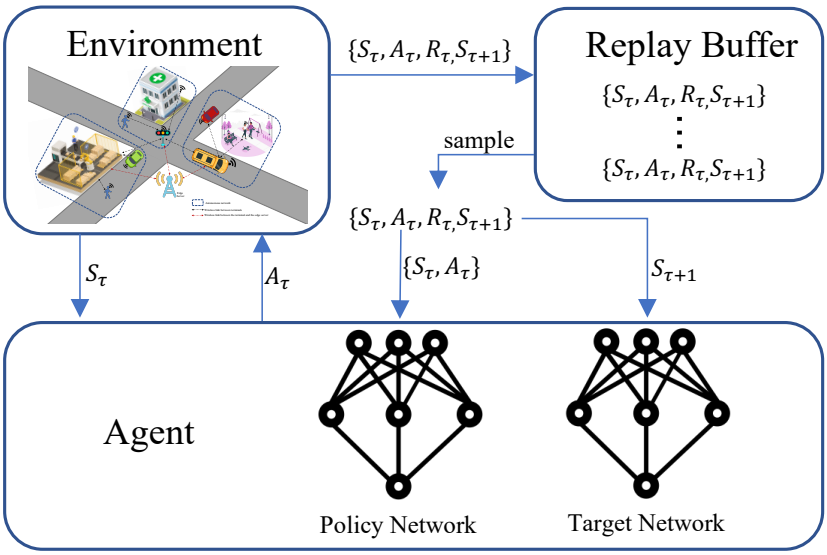


Figure 3. DQN architecture.

Algorithm 1: DQN-based dynamic offloading

1. **Input:** Environment configuration parameters;
2. **Output:** Offloading decision;
3. For each training episode $e = 1, 2, \ldots, E$ do
4. Observe the current environment state S_t;
5. Select action A_t using epsilon-greedy policy;
6. Calculate reward R_t and observe the new environment state S_{t+1};
7. Store the transition (S_t, A_t, R_t, S_{t+1}) in the replay buffer;
8. Sample a minibatch of transitions and update the policy network parameters using Equation (27) and the MSE loss with Adam optimizer;
9. Every fixed number of episodes, copy the policy network parameters to the target network;
10. **End for**

5. Analysis of the Simulation and Experimental Results

5.1. Semantic Communication Model Simulation Setup

The simulation environment is based on the Ubuntu 18.04 64-bit operating system, with hardware configured as an Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GHz processor and an NVIDIA P100 GPU. All deep learning models and experiments are implemented using the PyTorch framework. The training parameters are shown in Table 1, where the exponential decay rates of the moving average for controlling momentum and squared gradients are set to 0.95 and 0.99, respectively. The medical image dataset is obtained from a publicly available Kaggle dataset [26]. To enhance the model's generalization capability, several preprocessing techniques are applied, including resizing the images to 224 × 224, random horizontal flipping, random rotation, conversion to tensors, and pixel value normalization. For performance comparison, three other benchmark systems are considered:

- Conventional Method:This approach uses Joint Photographic Experts Group (JPEG) for image source coding and Low-Density Parity-Check (LDPC) codes for channel coding. Specifically, the coding follows the Quasi-Cyclic LDPC (QC-LDPC) codes of the IEEE 802.16e standard, with an information block length of 576 bits, a coding rate of 5/6, and Binary Phase-Shift Keying (BPSK) modulation.

- CNN-DeepSC: This method employs a CNN encoder to extract semantic information from the images. The CNN encoder consists of convolutional layers, batch normalization layers, ReLU activation layers, and pooling layers, which sequentially extract and refine image features, transforming low-level pixel information into high-level semantic features.
- MLP-DeepSC: This benchmark model is based on the MLPs model proposed for image processing tasks in [16], using a Vision Transformer as the semantic encoder at the transmitter. The Transformer encoder utilizes Multi-Head Self-Attention (MSA) and MLP modules to capture global features.
- K-DeepSC(fast): This model employs FastKAN, implemented in [27], to replace the KAN module in the K-DeepSC model, significantly improving computational efficiency. FastKAN uses radial basis functions with Gaussian kernels instead of the cubic B-spline basis functions used in the original KANs, thereby avoiding the complex processes of iteratively calculating B-spline basis functions and grid scaling.

Table 1. Parameter setting of K-DeepSC.

Parameters	Value
Batch size	16
Input size	224 × 224
Epoch	300
opt betas	0.95, 0.99
Learn rate	3×10^{-5}
Optimizer	AdamW optimizer
Train channel SNR	12 dB

Performance Analysis

To evaluate the performance of the proposed model in the image-reconstruction task, we conducted simulation tests under AWGN and Rayleigh channels at SNR levels ranging from −4 to 18 dB, focusing on its PSNR performance. Both the CNN-DeepSC and MLP-DeepSC models were trained at an SNR of 12 dB. The simulation results are shown in Figure 4, and the model parameter sizes are presented in Table 2. As shown in Figure 4, the proposed K-DeepSC and K-DeepSC(fast) models achieve comparable performance to MLP-DeepSC, while the required parameters are reduced by almost 44%. Although CNN-DeepSC has the least number of parameters, its limited ability to capture global information results in suboptimal performance. In contrast, under poor channel conditions (e.g., SNR < 4 dB), traditional syntax communication exhibits very low PSNR values, rendering effective image reconstruction impossible. However, the proposed K-DeepSC semantic communication system, with fewer model parameters, is less affected by the channel signal-to-noise ratio, maintaining a high PSNR even under extremely adverse channel conditions.

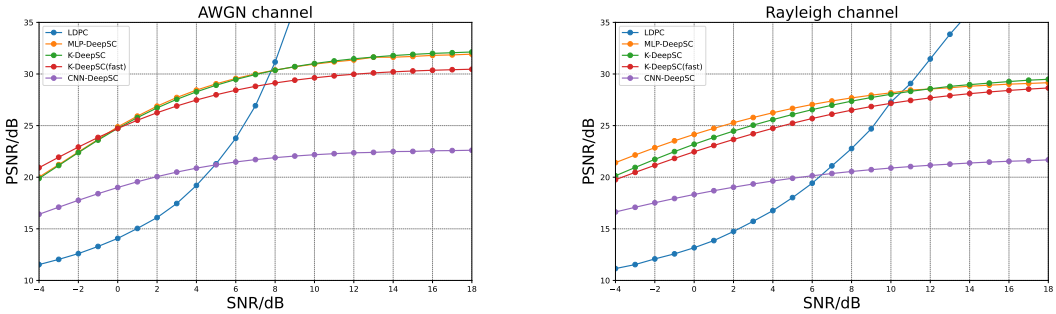


Figure 4. Performance comparison of K-DeepSC and four other systems under AWGN and Rayleigh Channels.

Table 2. Number of model parameter.

Model	Parameters
K-DeepSC	67.66 M
K-DeepSC(fast)	67.30 M
MLP-DeepSC	120.00 M
CNN-DeepSC	36.37 M

5.2. Terminal Semantic Task Offloading Simulation Environment Setup

This paper simulates a network environment of 400 m × 1200 m, which contains three adjacent ACPNs and an edge server. Each ACPN measures 400 m × 400 m, with a fixed PT located at the center of each area, responsible for sensing the current computational tasks in the environment and generating corresponding offloading decisions. It is assumed that there are 10 STs in the environment, which are randomly distributed within the areas, with their movement speeds varying in the range of [1, 10] m/s. The computing processing capability of the STs is between 20,000 and 25,000 Mcycles/s. To simplify the experimental process, the heterogeneity of computing terminals is not considered. Other communication environment parameters are shown in Table 3, while the DQN algorithm parameters are listed in Table 4.

Table 3. Communication environment parameter setting.

Parameters	Value
Primary Terminal Speed	0 m/s
Primary Terminal Transmission Power	50 dbm
Primary Terminal Bandwidth	20 MHZ
Secondary Terminal Speed	1–10 m/s
Secondary Terminal Transmission Power	10 dbm
Secondary Terminal Bandwidth	40 MHZ
Communication Noise Power	0.001 dbm
Fixed loss	30 dbm
Path loss	2

Table 4. DQN parameter setting.

Description	Value
Train Episode	1000
Activation Function	ReLU
Policy network learning rate	2×10^{-5}
Target network learning rate	2×10^{-5}
Number of hidden units per layer	256
Epsilon Start	0.95
Epsilon End	0.01
Gamma	0.95

5.2.1. Comparison of Different Parameters

Batch size and learning rate are crucial hyperparameters in DQN. Figure 5 illustrates the impact of different batch sizes and learning rates on the convergence of the DQN algorithm. From the figure, it can be observed that when the batch size is set to 256, the algorithm achieves a higher reward and begins to converge after approximately 100 episodes, while with a batch size of 128, convergence only starts around 200 episodes. When the learning rate is set to 0.002 and 0.0002, the convergence curves exhibit significant fluctuations, indicating that the learning rate is too high, leading to large updates in the network parameters and causing instability in the Q-value estimates. However, when the learning rate is set to 0.00002, the curve is relatively stable, and the algorithm successfully converges. Therefore, we set the batch size of DQN to 256 and the learning rate to 0.00002.

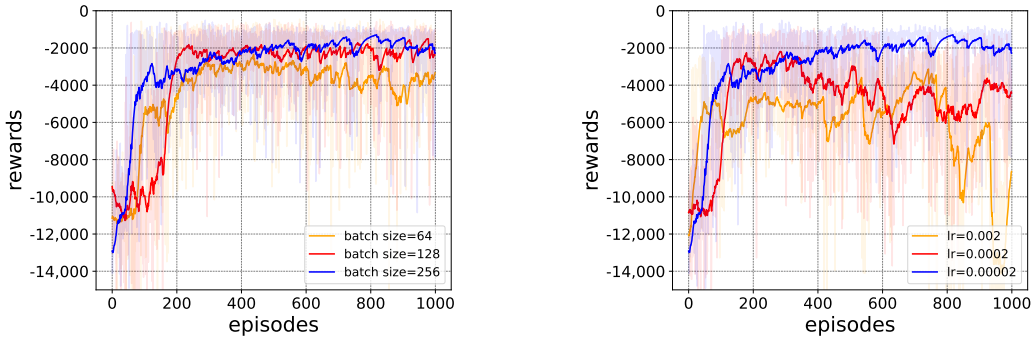


Figure 5. Convergence of the DQN algorithm under different batch size and learning rate.

5.2.2. Comparison of Different Algorithms

Figure 6 demonstrates the performance of different offloading strategies. From the figure, it can be seen that the greedy algorithm performs the worst, yielding the lowest rewards. This is because it only considers offloading computational tasks to the terminal with the least load, ignoring other critical factors such as the success rate of link communication, leading to a low task completion rate and high penalties. In contrast, the DQN algorithm outperforms the PPO algorithm in both rewards and convergence performance. This advantage is due to DQN's efficient utilization of experience replay and target network update mechanisms when handling discrete task decisions, thereby enhancing the stability and learning efficiency of the strategy. The rewards for fully offloading to the edge server are relatively stable, primarily benefiting from the powerful processing capabilities of the edge server and low queuing delays. However, due to the longer transmission distances and higher communication delays, its overall performance is inferior to that of the DQN algorithm.

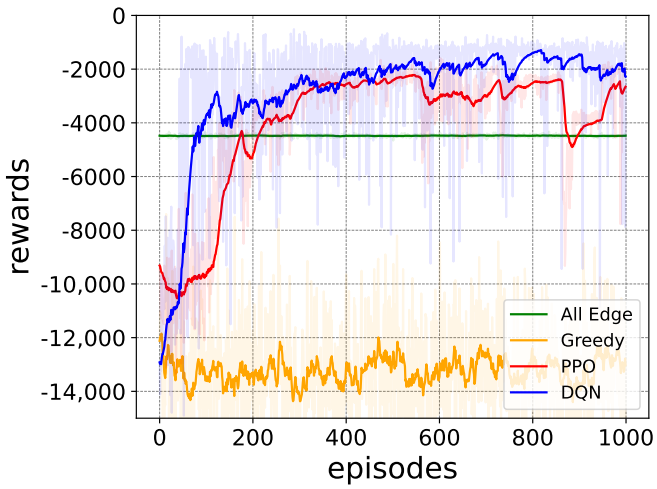


Figure 6. Comparison of rewards among different algorithms.

5.2.3. Impact of Different Task Sizes on Algorithms

Figure 7 illustrates the impact of task size on rewards and completion rates. From the figure, it can be observed that, except for the greedy algorithm, the rewards of other algorithms gradually decrease as task size increases. Task size changes have minimal impact on the greedy algorithm. This approach, which overlooks the reliability of inter-terminal communication and prioritizes offloading tasks to terminals with lower loads, ultimately results in decreased task completion rates. Consequently, this leads to penalties and lower rewards. In contrast, the rewards of the DQN algorithm outperform those of the PPO algorithm across different task sizes. In the scenario where all tasks are offloaded to the edge server, the rewards significantly decline with increasing task sizes. This is due to

the impact of task size on transmission times, which prevents meeting latency requirements and increases penalties. Figure 4 also shows the influence of different task sizes on task completion rates. Both the DQN and PPO algorithms maintain completion rates close to 1 across all task sizes, while the greedy algorithm's completion rate stabilizes around 0.8. In contrast, the scheme of offloading all tasks to the edge server experiences a sharp decline in completion rates for larger tasks, eventually maintaining only around 0.2.

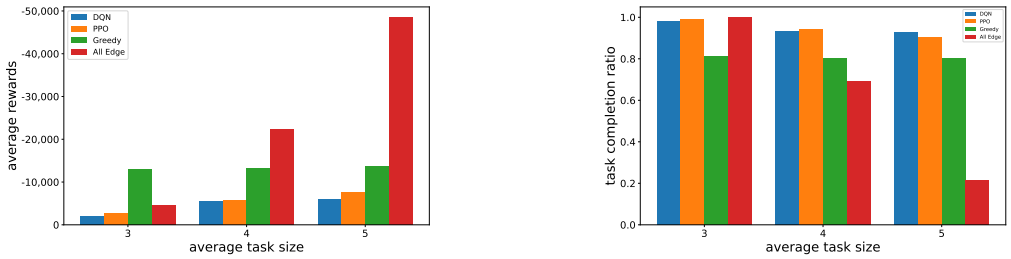


Figure 7. Comparison of rewards and task completion rates among different algorithms under different sizes of semantic tasks.

5.2.4. Impact of Different Numbers of Terminals on Rewards of Different Algorithms

Figure 8 illustrates the impact of the number of terminals on rewards. Since the variation in the number of terminals has no significant effect on the latency of offloading all tasks to the edge server, this comparison focuses only on the DQN, PPO, and Greedy algorithms. In the DQN algorithm, as the number of terminals increases, more tasks can be offloaded to the terminals for processing, thereby reducing the overall processing latency. The rewards of the PPO algorithm increase slightly with the increasing number of terminals, but its performance still lags behind that of the DQN algorithm. The greedy algorithm tends to offload tasks to terminals with lower loads. When the number of terminals is 9, the greedy algorithm offloads some tasks to the edge server, which improves the task completion rate and slightly increases its rewards. Meanwhile, when the number of terminals increases to 11, the reduction in the task queue leads to decreased queuing latency, resulting in a slight improvement in the rewards compared to when there are 10 terminals. Nevertheless, overall, the performance of the greedy algorithm still shows a significant gap compared to the DQN algorithm.

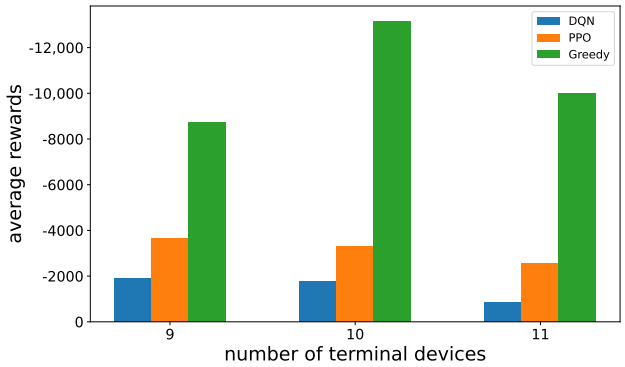


Figure 8. Rewards comparison of different algorithms under different number of terminals.

6. Conclusions

This paper addresses the challenges posed by limited storage and computing resources in the application of semantic communication at the network edge. We propose a terminal computing-enabled semantic communication solution, wherein the proposed K-DeepSC semantic communication model significantly reduces the storage resource requirements of terminals while maintaining high performance. Furthermore, to fully leverage terminal computing for processing semantic computing tasks, we explore the issue of computation

offloading in dynamic ACPNs, transforming this problem into an MDP and solving it using the DQN algorithm. Extensive simulation results demonstrate that our solution effectively reduces the latency of semantic task computation.

7. Limitations and Future Work

To the best of our knowledge, this paper is the first to address the issue of computation offloading in terminal-side computing power networks, aiming to lay the groundwork for further research in this field. This paper primarily considers the mobility of terminal devices in terminal-side computing power networks. However, many important issues remain to be explored. For example, the heterogeneity of computational capabilities among terminal devices may lead to significant differences in processing power and response time across devices, which in turn affects overall system performance. Therefore, efficient service deployment and task scheduling are crucial. Designing and implementing topology-aware and efficient service deployment algorithms for terminal-side computing power networks is a key challenge that needs to be addressed [21]. In addition, the sensitivity of terminal devices to energy consumption is another important concern. How to reduce energy consumption while maintaining computational performance remains an area for further research. Finally, given that terminal devices interact directly with users, ensuring the security and privacy of user data is of paramount importance. The use of semantic communication for data transmission offers an effective solution. In semantic communication systems, encoding and decoding neural networks are deployed at the sender and receiver, respectively, with different services corresponding to different network structures. As a result, even if an attacker intercepts the transmitted data, without knowledge of the target network structure, they would be unable to restore the data to a useful form.

Through experiments, we found that although K-DeepSC can achieve performance comparable to that of MLP-DeepSC with a larger parameter count when using a smaller parameter count, it consumes more computational resources during the training process. Additionally, as the parameter count increases, K-DeepSC faces the problem of gradient vanishing. The main reasons for these phenomena lie in the complexity of B-spline calculations and the fact that during the gradient descent training process, the gradient may drop to zero when the B-spline function approaches zero.

To address these issues, we referred to the method proposed in the literature [27] that uses radial basis functions to replace higher-order B-spline functions. This improvement effectively reduces the computational complexity of KANs while avoiding the gradient vanishing problem encountered in training B-spline functions. Compared to K-DeepSC, K-DeepSC(fast) reduces computational resource consumption by 37.01% while maintaining high performance. Although the computational resource consumption of K-DeepSC(fast) is still 31.34% higher than that of MLP-DeepSC, the literature [27] provides us with innovative ideas for improving KANs. Future research will focus on finding other simpler local kernel functions to replace B-spline activation functions in order to enhance the computational efficiency of KANs.

Author Contributions: Conceptualization: W.L. and Q.Z.; methodology: W.L. and Q.Z.; software: W.L.; investigation: Q.Z., L.L. and W.A.; resources: Q.Z., L.L. and W.A.; data curation: W.L. and Q.Z.; writing—original draft preparation: W.L. and Q.Z.; writing—review and editing: L.L. and W.A.; visualization: Q.Z. and W.A; funding acquisition: Q.Z., L.L. and W.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the Beijing Municipal Education Commission's Emerging Interdisciplinary Platform Construction—Digital Inkjet Printing Technology and Key Technology R&D Platform (No. 04190123001/003) and the Pediatric Teaching Audio-Visual Database Operational Agreement at Peking Union Medical College Hospital (No. 11000301281); and the Doctoral Research Start-up Fund for the research on Concept Learning Methods for Symbolic Block Data (No. 27170124012).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

1. Niu, K.; Dai, J.; Zhang, P.; Yao, S.; Wang, S. 6G-oriented Semantic Communications. *Mob. Commun.* **2021**, *45*, 85–90.
2. Zhang, P.; Xu, W.; Gao, H.; Niu, K.; Xu, X.; Qin, X.; Yuan, C.; Qin, Z.; Zhao, H.; Wei, J.; et al. Toward wisdom-evolutionary and primitive-concise 6G: A new paradigm of semantic communication networks. *Engineering* **2022**, *8*, 60–73. [CrossRef]
3. Zhang, P.; Niu, K.; Yao, S.; Dai, J. Semantic communications for future: Basic principle and implementation methodology. *J. Commun.* **2023**,*44*, 1–14.
4. Xie, H.; Qin, Z.; Li, G.Y.; Juang, B.H. Deep learning enabled semantic communication systems. *IEEE Trans. Signal Process.* **2021**, *69*, 2663–2675. [CrossRef]
5. Weng, Z.; Qin, Z. Semantic communication systems for speech transmission. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 2434–2444. [CrossRef]
6. Xie, H.; Qin, Z.; Tao, X.; Letaief, K.B. Task-oriented multi-user semantic communications. *IEEE J. Sel. Areas Commun.* **2022**, *40*, 2584–2597. [CrossRef]
7. Xie, H.; Qin, Z.; Li, G.Y. Task-oriented multi-user semantic communications for VQA. *IEEE Wireless Commun. Lett.* **2021**, *11*, 553–557. [CrossRef]
8. Peng, X.; Qin, Z.; Tao, X.; Lu, J.; Hanzo, L. A Robust Semantic Text Communication System. *IEEE Trans. Wireless Commun.* **2024**, *23*, 11372–11385. [CrossRef]
9. Fu, Q.; Xie, H.; Qin, Z.; Slabaugh, G.; Tao, X. Vector quantized semantic communication system. *IEEE Wirel. Commun. Lett.* **2023**, *12*, 982–986. [CrossRef]
10. Xie, H.; Qin, Z.; Li, G.Y. Semantic communication with memory. *IEEE J. Sel. Areas Commun.* **2023**, *41*, 2658–2669. [CrossRef]
11. Weng, Z.; Qin, Z.; Tao, X.; Pan, C.; Liu, G.; Li, G.Y. Deep learning enabled semantic communications with speech recognition and synthesis. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 6227–6240. [CrossRef]
12. Xiao, W.; Tang, Y.; Liu, J.; Wu, D.; Alzahrani, B.; Hao, Y.; Zhou, N. Semantic-driven efficient service network towards smart healthcare system in intelligent fabric. *IEEE Trans. Netw. Sci. Eng.* **2022**, *10*, 2480–2489. [CrossRef]
13. Xie, H.; Qin, Z. A lite distributed semantic communication system for Internet of Things. *IEEE J. Sel. Areas Commun.* **2020**, *39*, 142–153. [CrossRef]
14. Song, M.; Ma, N.; Dong, C.; Xu, X.; Zhang, P. Deep Joint Source-Channel Coding for Wireless Image Transmission with Adaptive Models. *Electronics* **2023**, *12*, 4637. [CrossRef]
15. Rakhmanov, A.; Wiseman, Y. Compression of GNSS Data with the Aim of Speeding up Communication to Autonomous Vehicles. *Remote Sens.* **2023**, *15*, 2165. [CrossRef]
16. Zhang, G.; Hu, Q.; Qin, Z.; Cai, Y.; Yu, G.; Tao, X. A unified multi-task semantic communication system for multimodal data. *IEEE Trans. Commun.* **2024**, *72*, 4101–4116. [CrossRef]
17. Lei, B.; Zhao, Q.; Zhao, H. Overview of Edge Computing and Computing Power Network. *ZTE Commun.* **2021**, *27*, 3–6.
18. Tang, Q.; Xie, R.; Feng, L.; Yu, F.R.; Chen, T.; Zhang, R.; Huang, T. Siats: A service intent-aware task scheduling framework for computing power networks. *IEEE Netw.* **2023**, *38*, 233–240. [CrossRef]
19. Zheng, D.; Luo, T.; Qiu, C.; Zhang, C.; Liu, Z.; Wang, X.; Jiang, J. In-network pooling: Contribution-aware allocation optimization for computing power network in B5G/6G era. *IEEE Trans. Netw. Sci. Eng.* **2022**, *10*, 1190–1202.
20. Zhou, X.; Jia, Q.; Hu, Y.; Guo, K.; Ma, Q.; Liu, H.; Xie, R. Autonomous computing and network convergence: Architecture, technologies, and prospects. *Chin. J. Internet Things* **2023**, *7*, 1–12.
21. Zhang, X.; Qu, Z.; Sun, Y.; Zhang, J.; Wang, W. Terminal-Side Computing Power Network: Architecture, and Key Technologies. *J. Beijing Univ. Posts Telecommun.* **2024**, *47*, 1.
22. Qin, Z.; Ying, J.; Yang, D.; Wang, H.; Tao, X. Computing networks enabled semantic communications. *IEEE Netw.* **2024**, *38*, 122–131. [CrossRef]
23. Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T.Y.; Tegmark, M. Kan: Kolmogorov-Arnold networks. *arXiv* **2024**, arXiv:2404.19756.
24. Han, D.; Pan, X.; Han, Y.; Song, S.; Huang, G. Flatten transformer: Vision transformer using focused linear attention. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 1–6 October 2023; pp. 5961–5971.
25. Li, B.; Hu, Y.; Nie, X.; Han, C.; Jiang, X.; Guo, T.; Liu, L. Dropkey for vision transformer. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 18–22 June 2023; pp. 22700–22709.

26. Kaggle. COVID-19 Image Dataset. Available online: https://www.kaggle.com/datasets/pranavraikokte/covid19-image-dataset/data (accessed on 25 July 2024).
27. Li, Z. Kolmogorov-arnold networks are radial basis function networks. *arXiv* **2024**, arXiv:2405.06721.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.