



## UvA-DARE (Digital Academic Repository)

### Adaptive Services Function Chain Orchestration for Digital Health Twin Use Cases

*Heuristic-boosted Q-Learning Approach*

Alsayed Kassem, J.; Zhong, L.; Taal, A.; Grosso, P.

#### DOI

[10.1109/NetSoft57336.2023.10175506](https://doi.org/10.1109/NetSoft57336.2023.10175506)

#### Publication date

2023

#### Document Version

Final published version

#### Published in

2023 IEEE 9th International Conference on Network Softwarization (NetSoft 2023) : proceedings

#### License

Article 25fa Dutch Copyright Act (<https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment>)

[Link to publication](#)

#### Citation for published version (APA):

Alsayed Kassem, J., Zhong, L., Taal, A., & Grosso, P. (2023). Adaptive Services Function Chain Orchestration for Digital Health Twin Use Cases: Heuristic-boosted Q-Learning Approach. In C. J. Bernardos, B. Martini, E. Rojas, F. L. Verdi, Z. Zhu, E. Oki, & H. Parzyjeglá (Eds.), *2023 IEEE 9th International Conference on Network Softwarization (NetSoft 2023) : proceedings: Boosting Future Networks through Advanced Softwarization : 19-23 June 2023, Madrid, Spain* (pp. 187-191). IEEE. <https://doi.org/10.1109/NetSoft57336.2023.10175506>

#### General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

#### Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible. University of Amsterdam (<https://dare.uva.nl>)

# Adaptive Services Function Chain Orchestration For Digital Health Twin Use Cases: Heuristic-boosted Q-Learning Approach

1<sup>st</sup> Jamila Alsayed Kassem  
MNS Lab, IvI  
University of Amsterdam  
Amsterdam, Netherlands  
j.alsayedkassam@uva.nl

2<sup>nd</sup> Li Zhong  
SNE, IvI  
University of Amsterdam  
Amsterdam, Netherlands  
li.zhong@student.uva.nl

3<sup>rd</sup> Arie Taal  
MNS, IvI  
University of Amsterdam  
Amsterdam, Netherlands  
a.taal9@upcmail.nl

4<sup>th</sup> Paola Grosso  
MNS, IvI  
University of Amsterdam  
Amsterdam, Netherlands  
p.grosso@uva.nl

**Abstract**—Digital Twin (DT) is a prominent technology to utilise and deploy within the healthcare sector. Yet, the main challenges facing such applications are: strict health data-sharing policies, high-performance network requirements, and possible infrastructure resource limitations. In this paper, we address all the challenges by provisioning adaptive Virtual Network Functions (VNFs) to enforce security policies associated with different data-sharing scenarios. We define a Cloud-Native Network orchestrator on top of a multi-node cluster mesh infrastructure for flexible and dynamic container scheduling. The proposed framework considers the intended data-sharing use case, the policies associated, and infrastructure configurations, then provisions Service Function Chaining (SFC) and provides routing configurations accordingly with little to no human intervention. As a result, we provide an adaptive network orchestration for digital health twin use cases, that is policy-aware, requirements-aware, and resource-aware.

**Index Terms**—Virtual Network Function, Programmable Infrastructures, Network Policy, Service Function Chains, Digital Health Twin, Heuristic

## I. DIGITAL HEALTH TWIN: THE CONCEPT

The concept of Digital Twins (DTs) is not new, and it originally appeared in the early 1990s [1] under different terminologies, such as the "Mirror Space Model" [2], "Information Mirror Mode" [3], etc. Although DT has been primarily discussed within engineering and industrial contexts, medical and health use cases are not excluded from the DTs' impact. Deploying a Digital Health Twin (DHT) utilises medical data-sharing and patient-generated data to empower personalised medicine. With that in mind, DHTs come as a natural, complementary approach to implementing personalised medicine, since it offers the capacity to model a distinct patient. In the EPI project<sup>1</sup>, we develop a framework to combine data analytics, and health decision support algorithms to create personalised insights for prevention, management, and intervention to providers and patients.

The adoption of DHTs is accompanied and accelerated by maturing and growing computing technologies. This is led by

The EPI project is funded by the Dutch Science Foundation in the Commit2Data program.

<sup>1</sup><https://enablingpersonalizedinterventions.nl/>

cloud computing and network virtualisation, which offers the means to facilitate knowledge discovery by provisioning on-demand computing and network resources. It is evident that one of the main contributing factors toward the successful and reliable deployment of DHTs is the underlying network paradigm connecting all the data-sharing components to effectively run a use case [4]. It is crucial for the network to abide by a set of requirements while running a DHT use case: policy-wise, latency-wise, and resource-wise. We address the challenges of adaptively provisioning SFC according to the use case's requirements, by automating the placement and assignment of microservices with three different approaches: Greedy heuristic, Deep Q-Learning (DQL), and Heuristic-boosted Deep Q-Learning (HDQL).

## II. RELATED WORK

SFC and VNF provisioning problems have been formalised and addressed by employing heuristics or DQL in the past. Recently, [5] formalised TO-DG heuristic-based approach to maximise the network throughput, while considering resource overhead. They consider the required CPU consumption of a VNF, link capacity, and maximum tolerable delay to search for optimal provisioning decisions. [6] takes a different approach and utilised DQL neural networks to outperform linear programming approaches in an effort to solve the SFC resource allocation problem. Similarly, the authors in [7] use RL-based techniques to formalise the same problem as an MDP and address it with a policy gradient learning agent. We add to the current literature three main contributions: 1) We specifically reason about the type of SFC that is being provisioned to enforce network security policies in DHT use cases, 2) We propose HDQL techniques to guide and facilitate the learning process according to prior knowledge of profiled data and can outperform a traditional DQL, 3) We add dynamic constraints to the provisioning problem; dynamic Network Points of Placement (N-PoP) candidate and prioritising different metrics with different use cases.

### III. HEALTH DATA SHARING POLICIES

In [8], we defined the collaboration logic model which the EPI data-sharing framework follows to aggregate higher-level data-sharing agreements, with lower-level network security goals to establish a policy-abiding data-sharing session. Based on the discussion with the hospitals within the EPI consortium, we can list the following security goals as: 1) Providing access control to the data resources, 2) Identifying and authenticating parties, 3) Health data integrity, confidentiality, and 4) Non-repudiation. These security goals can be achieved by applying different security mechanisms, such as deploying access control and security protocols namely, SSL, SSH, IPsec, firewalling, and the security gateway systems [9].

We build on top of previous work [8] [10], and we aim to deploy and provision on-the-fly exemplary reliable Virtual Network Functions (VNF), which we call Bridging Functions (BFs), that can accomplish a set of security goals. The framework's goal is to define an adaptive BFs Chain (BFC) orchestrator, enforcing defined network policies. We provide an access control mechanism by containerising a ready-to-deploy firewall function, and we address the rest by implementing standard security protocols to encrypt traffic. Once the security goal is specified, then we can map that to the mandated network services, and to the defined enforcement primitives: *Filter* traffic (F) and/or *transform* traffic (T). The framework dynamically provisions these services by placing the BFs on available N-PoPs, assigning the service requests to the running function (BF), and routing traffic along the function's chain to enforce a policy. Along with the data-sharing policy changes, the available N-PoPs are constrained with different use cases running. As a result, the high-level policy is defined and specified by two 3-tuples:  $\langle actors, acts, inRelation \rangle$ , and  $\langle endNodes, BFC, N-PoPs \rangle$ , where the second one is in accordance to the lower-level network policy.

### IV. DHT USE CASES AND N-POP RESTRICTIONS

We define a number of data-sharing use cases to effectively utilise patients-generated data within the EPI consortium.

**Electronic Health Records Repository (EHR):** Remote access to a patient's medical history is the first step towards a DHT model. This use case is built to run EHR data-sharing scenarios where there are two N-PoPs affiliated with healthcare institutions (HI), and an isolated third-party research centre N-PoP. Fig. 1 shows the use cases' configuration and the infrastructure setup, where the data-sharing movement is expected from a remote user network to HI network and vice versa. The third-party research centre is uninvolved in this transaction, and hence the affiliated third N-PoP is isolated from the rest. The placement algorithm will not consider it while placing the BFC request and will place the functions on the other two to secure network traffic according to the BFC request.

**Machine Learning model sharing:** We need to accelerate and support the deployment of ML-featured applications (such as psychiatry diagnosis, effective drug prescriptions, and side effects predictions, etc.), we define this use case where ML and analytics algorithms are sent from a research center

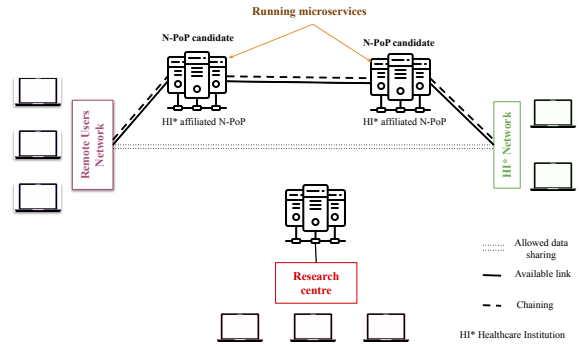


Fig. 1: The infrastructure graph configuration under the EHR and the Streaming use cases.

to be trained on data residing in the HI. Moreover, data movement is allowed again from the data provider (the HI) back to the algorithm provider (the research centre), so that the distributively trained model can be joined back again into one (more accurate) model, and shipped back to be ready to use. As a result, the policy dictates the availability of links across affiliated N-PoPs to ensure who and where data is handled, as illustrated in Fig. 2. After establishing those restrictions on traffic flow, we then reconfigure the network to adapt to that, and secure the network traffic.

**Healthcare data streaming:** This use case describes health data streaming, an example application is to monitor the patient's status via wearable data, provide timely interventions, etc. The data involved in this use case is sensitive data, which means that similarly as in use case A, HI-affiliated N-PoPs are only considered for placement.

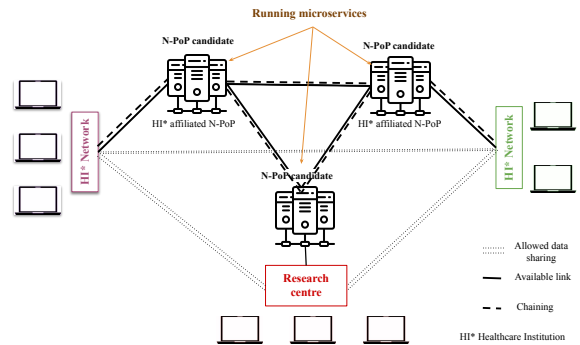


Fig. 2: The infrastructure graph configuration under the ML-model sharing use case.

### V. PROVISIONING APPROACHES

#### A. Greedy Heuristic BFC Deployment Algorithm

We first deploy a greedy-based heuristic approach: this reduces complexity with increasingly complex network policies and use cases, while it provides a manageable best-effort provisioning decision by choosing the first N-PoP candidate

meeting the placement constraints. This approach depends on accurate CPU profiles and does not react to resource usage and network latency bursts and anomalies. It is not concerned with the most optimal provisioning choice, but provides a working one.

### B. DQL Algorithm

The performance of the first approach is highly dependent on the accuracy of the CPU and delays profiles. Compared to traditional heuristic-based resource scaling methods, Reinforcement Learning-based (RL) solutions are equipped to deal un-profiled network and resource bursts, instead this DQL-approach relies on querying the current state and reacting via provisioning actions to maximise performance rewards.

**Action space:** The discrete actions of microservices placing and assignment are structured as [Cluster ID, Place/ Map/ Destroy, Instance ID, Microservice ID, Proxy ID]. The first value specifies the cluster that is considered, *i.e.* the N-PoP candidate. The second value specifies the type of provisioning action: placing a new instance, mapping a running instance to a request, or deleting an idle microservice instance. The third and fourth values, respectively, refer to the microservice instance and the type of microservice. Lastly, the Proxy ID identifies the proxy we are configuring to chain the instantiated microservices and handles the network services requests.

**State Space:** The state space consists of the infrastructure's variables, which the DQL interacts with to monitor the environment's changes based on decisions. The provisioning is done across multiple sites, and the DQL algorithm needs to make a satisfactory trade-off between resource cost and latency. Therefore, the state of the policy learning agent should contain information about: CPU utilisation/cluster, placement status of microservices, the number of microservices across the clusters, and the response time of requests.

**Reward Function and policy:** The reward function measures the performance incentive for the agent to perform a new action, based on the infrastructure's current state. The entire reward  $R_{all}$  at time  $t$  is calculated as the weighted sum of reward in resource cost  $R_{res}$  and reward in performance  $R_{perf}$ , which is shown in:

$$R_{all} = \alpha R_{res} + \beta R_{perf} \quad (1)$$

Hyperparameters  $\alpha$  and  $\beta$  are used to control the importance of these two values compared to the entire reward. One effective definition of reward function steers the agent towards better performance with higher utilisation of resources. In the traditional DQL approach, the policy is set to map an observable state  $s_t$  to a provisioning action  $a_t$  at a time  $t$ . The policy is optimised by learning the Q-value performing  $a_t$  in state  $s_t$ .

### C. Heuristic-boosted DQL Algorithm

In an effort to handle both profiled and unprofiled workload of BFs, we deploy the HDQL approach. While the previous tool can reactively adapt the network's configuration to optimally provision BFC requests, it still can take a long time to

converge with increasingly complex requests, use cases, and N-PoP configurations. Subsequently, we propose to combine both provision approaches to deploy a Heuristic-boosted DQL provisioning of SFC. We aim to accelerate the decision-making, and guide the model learning via a new action policy [11].

## VI. EXPERIMENTS AND RESULTS

The security enforcement primitives are consistent throughout all the use cases, such that the *firewall* function is mandated to provide access control to incoming traffic towards the healthcare institution. Likewise, all outgoing traffic should be *encrypted* to protect sensitive data. This logic is applied to all traffic, in the use cases. The EHR use case is profiled to be a small load application with an expected average send rate of 100-200 kB/s. The ML-model sharing use case is defined to be, also, a small load use case with an expected (average) periodic send rate of 100-200 kB/s. Additionally, the streaming use case is profiled to be a large load use case with a 1-3 MB/s send rate. Accordingly, the CPU profiles of different BFCs under different use cases' send rates are recorded in [12], and further used via the heuristic algorithm to make provisioning decisions. Moreover, the provisioning algorithm should comply with multiple use case requirements. Maximum CPU usage is prioritised with the EHR and ML-model sharing use cases, compared to the streaming use case where low latency overhead is required as well.

To evaluate placement and assignment decisions taken by the different provisioning tools, we run the three use cases according to the specified send rate workloads, and associated CPU profiles. We first reconfigure the network graph as Kubernetes clusters, illustrated in Fig. 1 and 2. The clusters are connected to form a cluster mesh via a cilium backend server<sup>2</sup>, to make microservices findable and routable across clusters. We set up the three clusters with similar resource pools using 12 VMs (Debian 2 cores, 2 GB of RAM, and 20 GB of storage), each cluster with three worker nodes and a master node for on site service orchestration. Cluster 1 and 2 refers to the running HI-affiliated N-PoPs, cluster 3 is the research centre (described in Sec. IV).

We evaluate the decisions made by different approaches by inspecting new instance vs CPU utilisation of running instances. We increase the number of concurrent clients utilising the BFC with locust<sup>3</sup>, and we record the instantiated new instances/microservice across all the clusters under the different use case configurations, with the growing number of clients. For replication purposes, the implementation and configurations of the experiments are available on GitHub<sup>4</sup>. We compare the number of pods (instances) to the actual utilized CPU per instance, to evaluate the CPU resource wastage, and further relate that to the latency.

Similarly, we evaluate the network performance by increasing the number of concurrent clients running different use

<sup>2</sup><https://cilium.io/>

<sup>3</sup><https://locust.io/>

<sup>4</sup><https://github.com/epi-project/Netsoft2023>

cases, and record the latency of processing one request (sending a request, and receiving a reply back). Low latency can be accomplished by providing high-performance networking, but the goal is to evaluate the overhead latency caused by adding network services in between end nodes, and the effect of different provisioning decisions. We try to minimise the delay when deploying the network function request, and ideally the latency overhead  $\approx 0ms$ . We collect the latency overhead by calculating the average of 10 queries, then measuring the effect of the chain addition compared to no functions in between.

**Results:** The complete results can be found in the longer paper version [13]. Here we report the most interesting findings, and we start by analysing the DQL and HDQL placement under the EHR use case. Reportedly, the N-PoPs constraints are that no microservice instances are placed on cluster 3 *i.e.* the Research centre cluster (or, if already running, not assigned to the requests running EHR traffic), as shown in Fig. 3. The provisioning decision, however, differs with each method, to capture the difference we collect the number of running pods /clusters and the effect of increasing the number of clients from 1 to 50 concurrently running. Moreover, the provisioning decisions are based on factors: CPU availability, and latency overhead. Hence, we also showcase the average CPU utilisation percentage/pod with each iteration, and the latency recorded to successfully resolve one request.

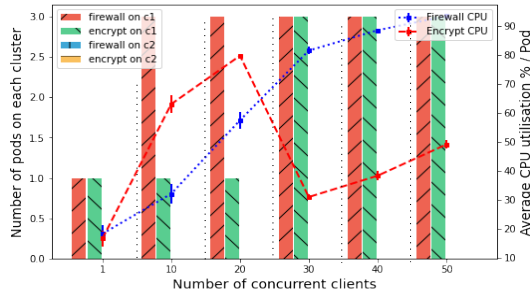


Fig. 3: The placement of microservices with DQL and HDQL algorithms and the average CPU utilisation running the EHR use case .

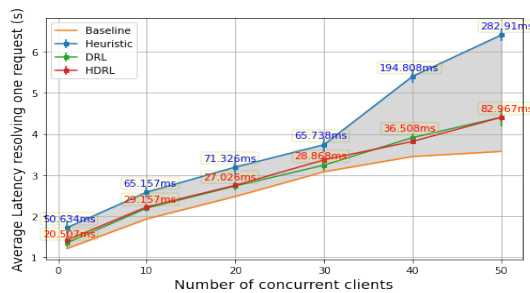


Fig. 4: Latency average while running the streaming use case under different placement methods, and the overhead compared to proxying traffic without passing through extra microservices.

We see that the placement starts by placing one replica (pod) of the firewall and encryption microservices both on cluster 1, with an average CPU utilisation of 20-25% running 1 client on both. The firewall utilisation ramps up faster than encryption to reach 65% with 10 concurrent clients, the DQL and HDQL placements preemptively act on this increase by adding 2 firewall pods running on cluster 1. With this increase, the utilisation of the firewall drops to 30% with 30 concurrent clients. Similarly, the encryption pods were increased after the utilisation reached 85% with 20 concurrent clients. With this approach, two main decisions to optimise placement are taken, the first is to place the encryption microservice on cluster 1, and by that only using the Kubernetes backend discovery service once, such that microservices on the same cluster belong to the same private network. Secondly, the DQL is more adaptive to resource bursts, with no need for profiling, and relying on accurate profiles of CPU usage. The HDQL approach behaves similarly and we get the same latency (Fig. 4). Here, the hyperparameters are set in the action policy. With this use case, latency overhead is not crucial, and hence occasional under-provisioning of resources is not an issue, and the heuristic placement is a sufficient tool for this use case.

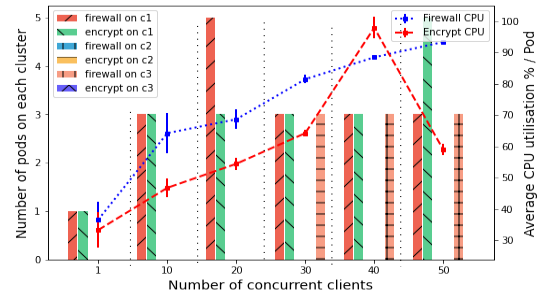


Fig. 5: The placement of microservices with DQL and HDQL algorithms and the average CPU utilisation running the ML-model sharing use case.

Similarly, the ML-model sharing use case has initially similar placements, the difference is that the utilisation is doubled indicating that the pods are assigned to handle double requests (due to the use case’s traffic flow). As shown in Fig. 5, the optimisation decisions taken by the DQL and HDQL approaches are demonstrated by deciding to place the firewall and encryption microservices on the same cluster c1. The number of instances of the firewall increased from 1 to 5 running 1 and 20 clients, respectively. The encryption’s instances also increased to 5 with 50 clients, and that to accommodate the increase of the average CPU utilisation. Then the algorithms start new instances of the firewall cluster 3, and the decision related to the lower round-trip latency, compared to cluster 2.

With the last use case, the healthcare data streaming use case, we prioritise latency, and there are three different placement decisions taken. First, the heuristic performs poorly with this use case in accordance with the latency overhead as shown in Fig. 4, where the chaining is still distributed, and

assignment decisions require two different lookups. With the DQL approach, the placement is distributed across clusters, but the chaining is on one cluster. With that, Kubernetes backend microservice discovery is optimised. With this experiment, due to the hyperparameters change, we end up wasting resources with the DQL approach as shown in Fig. 6(b). Although with these provisioning decisions, we provide the best latency overhead, there exist better actions and hence we have a third different placement with HDQL. The HDQL provisioning provides approximately equal latency with maximal CPU utilisation (shown in Fig. 6(c) and 4. As a result, the heuristic-based approach proved to be sufficient with the EHR and ML-model sharing use cases, where we end up under-provisioning but inflicting tolerable latency. That is especially true since low latency overhead is not crucial in running said use cases, instead, we prioritise minimal CPU wastage. The HDQL tool performs the best while running the streaming use case, where minimal overhead latency was achieved, but with seemingly no over-provisioning and resource wastage.

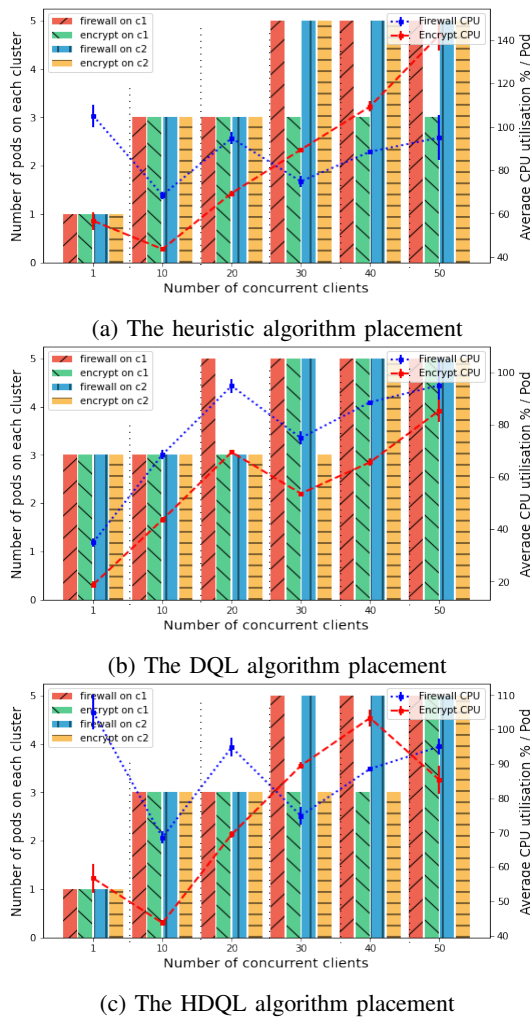


Fig. 6: The placement of microservices and the average CPU utilisation running the streaming use case.

## VII. CONCLUSION

To run DHT use cases, we must first consider data-sharing policies (including network policies); translate them into actionable service function chain requests, *i.e.* BFCs. The provisioning of a BFC, that is a new instance placement and/or assignment of an incoming request to an old running instance, depends on the use case's requirements and the state of the infrastructure. We prioritise latency or/and minimising resource wastage, and we do that by modelling the decision as a constrained optimising problem. Initially, we address the requirements and constraints via heuristic, which can query the infrastructure state and output decisions influenced by the BFC CPU profiles. Next, we propose DQL methods, which provide more resilience to un-profiled bursts and network degradation. Finally, we combine both approaches into a solution that accomplishes lower latency overhead, with minimal CPU wastage. The framework we propose can be used within any general context, and it effectively provisions network resources to deploy DHT use cases. We conclude that heuristic-based approaches are sufficient when the latency overhead is not crucial, while HDQL tools are most effective otherwise.

## REFERENCES

- [1] David Hillel Gelernter, *Mirror World: Or the day software puts the universe in a shoebox: How it will happen and what it will mean*, Oxford Univ. Press, 1991.
- [2] M. Grieves, *Product Lifecycle Management: driving the next generation of Lean Thinking*, McGraw-Hill, 2006.
- [3] Michael W. Grieves, "Product lifecycle management: The new paradigm for enterprises," *International Journal of Product Development*, vol. 2, no. 1/2, pp. 71, 2005.
- [4] Maggie Mashaly, "Connecting the twins: A review on digital twin technology its networking requirements," *Procedia Computer Science*, vol. 184, pp. 299–305, 2021.
- [5] Yi Yue, Bo Cheng, Meng Wang, Biyi Li, Xuan Liu, and Junliang Chen, "Throughput optimization and delay guarantee vnf placement for mapping sfc requests in nfv-enabled networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4247–4262, 2021.
- [6] Tom Jenno Wassing, Danny De Vleeschauwer, and Chrysa Papagianni, "A machine learning approach for service function chain embedding in cloud datacenter networks," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, 2021, pp. 26–32.
- [7] Jaehoon Koo, Veena B. Mendiratta, Muntasir Raihan Rahman, and Anwar Walid, "Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics," 2019.
- [8] Jamila Alsayed Kassem, Cees De Laat, Arie Taal, and Paola Grosso, "The epi framework: A dynamic data sharing framework for healthcare use cases," *IEEE Access*, vol. 8, pp. 179909–179920, 2020.
- [9] Romain Laborde, Michel Kamel, François Barrère, and Abdelmalek Benzekri, "Implementation of a formal security policy refinement process in wbm architecture," *Journal of Network and Systems Management*, vol. 15, no. 2, pp. 241–266, 2007.
- [10] Christopher A. Esterhuysen, Tim Müller, L. Thomas Van Binsbergen, and Adam S. Z. Belloum, "Exploring the enforcement of private, dynamic policies on medical workflow execution," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 481–486.
- [11] Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan, "Heuristic-guided reinforcement learning," *CoRR*, vol. abs/2106.02757, 2021.
- [12] Jamila Alsayed Kassem, Adam Belloum, Tim Müller, and Paola Grosso, "Utilisation profiles of bridging function chain for healthcare use cases," in *2022 IEEE 18th International Conference on e-Science (e-Science)*, 2022, pp. 475–480.
- [13] Jamila Alsayed Kassem, Li Zhong, Arie Taal, and Paola Grosso, "Adaptive services function chain orchestration for digital health twin use cases: Heuristic-boosted q-learning approach," 2023.