2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress

MobileDLSearch: Ontology-based Mobile Platform for Effective Sharing and Reuse of Deep Learning Models

Zhangcheng Qiang*†, Yuxin Zhang*, Pari Delir Haghighi*,
Abdur Rahim Mohammad Forkan‡, Prem Prakash Jayaraman‡, Junyao Deng*
*Monash University, Australia
†Australian National University, Australia
‡Swinburne University of Technology, Australia

***Abstract*—Recently, on-device inference using deep learning (DL) models for mobile and edge devices has attracted significant attention in ubiquitous computing due to its lower latency, better performance and increased data privacy. Adopting pre-trained DL models as the backbone for downstream tasks has become the consensus of the Artificial Intelligence (AI) community since it can remarkably accelerate the DL deployment process. However, most of the pre-trained DL models are not suitable for resource-constraint platforms. Further, there is a scarcity of platforms providing a unified way to store, query, share and reuse pre-trained DL models, especially for mobile applications. To address these limitations, this paper proposes an ontology-based platform (MobileDLSearch) that offers end-users greater flexibility to store, query, share and reuse pre-trained DL models for various mobile applications. The proposed MobileDLSearch uses a standardised ontology to represent various DL models with different backends (e.g., TensorFlow, Keras and PyTorch), and provides an intuitive and interactive user interface to support search and retrieval of DL models. It also implements an automatic model converter to optimise desktop/laboratory-oriented pre-trained DL models for mobile platforms, and has an on-device real-time model integration module to benchmark the model's performance on mobile devices. The evaluation results demonstrate the usability of the proposed MobileDLSearch to help end-users quickly search, deploy and benchmark DL models for various on-device inference tasks.**

***Index Terms*—Deep Learning, Ontology, Mobile Platform, Edge Device, Semantic Search**

I. Introduction

Deep Learning (DL) has gained increasing attention in recent years due to its ability to produce better accuracy than traditional machine learning (ML) techniques for complex problems [1]. The rapid development of mobile computing offers the potential to run DL models on mobile and edge devices. While DL is performed on mobile devices, it requires less data transmission and network communication costs. Compared with conventional cloud computing paradigms, on-device DL can significantly reduce latency and improve data privacy [2]. Also, there are some improvements made in mobile DL, including mobile-based DL frameworks (e.g., TensorFlow Lite, PyTorch Mobile), optimisation algorithms (e.g., quantisation, pruning and compression) and hardware (e.g., mobile graphic processing units). These techniques have significantly improved the performance of on-device DL inference, including accuracy, speed and power consumption [3]. However, deploying DL models on mobile devices is still not an easy task. Limited DL models are designed for mobile devices and these models are insufficient to handle the miscellaneous data collected from dynamic mobile computing environments. Therefore, building mobile DL models from scratch, creating new data is time-consuming and resource-intensive. A typical approach is reusing the pre-trained models similar to the application, fine-tuning these models on domain-specific data and leveraging transfer learning if necessary [4]. For example, a DL model originally developed to recognise human activity for a diabetic application (to determine the amount of time a person is performing certain exercises) can be re-purposed for another healthcare application that is detecting activity for back pain related application with a relatively slight transfer learning.

Although reusing the pre-trained models could significantly speed up deploying the DL models on mobile devices, we notice several challenges arise in practice. Firstly, finding an appropriate DL model from a variety of model zoos is difficult for non-expert users. DL models vary in their layer structures and parameter settings, and libraries such as TensorFlow, Keras and PyTorch use different syntaxes to develop and represent DL models. There is a lack of common and well-agreed standards for representing DL models [5]. Secondly, most of the DL models are developed for desktop computer platforms or laboratory settings, and only a small number of them have mobile versions. Desktop/laboratory-based models cannot be reused directly by mobile devices, and thus model conversion becomes an essential requirement. Currently, state-of-art libraries only provide limited support for this function [6]. Thirdly, on-device benchmarking is critical for the model selection because the converted mobile lite versions may lose accuracy on users' mobile devices. The loss of accuracy is caused by factors such as resource-constraint mobile devices and user-specific mobile environments [7]. An all-in-one platform that allows users to store, query, share and reuse the DL models on mobile devices can greatly facilitate deploying DL models on mobile applications and pave a way

978-1-6654-1762-4/21/$31.00 ©2021 IEEE
DOI 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics53846.2021.00023

to make mobile DL more accessible to wider communities.

This paper proposes MobileDLSearch, an ontology-based mobile platform for effective sharing and reuse of DL models. Apart from filling the knowledge gap in limited DL sharing platforms specified on the mobile application domain, we also tackle the key challenges of searching and reusing the pre-trained DL models from the enormous desktop/laboratory-based model zoos. In doing so, we aim to help DL engineers save a great deal of effort in finding appropriate DL models based on their own applications. The novelty of the proposed platform is its ability to use semantic technologies to represent DL models. The proposition of using ontologies enables users to query DL models stored in the platform at different levels of granularity. For example, a user can query a DL model based on the application domain and/or related DL model performance metrics such as accuracy and/or DL model type, number of layers, layer type, etc. The platform is the first attempt towards using ontologies for describing DL models to facilitate and promote sharing and reuse of DL models that would benefit both novices and experienced DL engineers. Also, we integrate the DL ontology with a unified mobile DL model converter and a generic on-device benchmarking user interface to improve the usability of the platform. By using our mobile platform, users can search DL models by certain criteria, view model details at different levels of granularity, perform real-time on-device model benchmarking and download/reuse auto-generated mobile compatible DL models within one mobile application. Specifically, MobileDLSearch proposed in the paper makes the following contributions:

- Designs and incorporates a novel DL ontology to represent and define various DL models in a standard and unified way.
- Develops and implements a mobile platform with an intuitive and interactive user interface that enables end-users to search for pre-trained DL models, view the model's low/high-level details and directly evaluate the model's performance on mobile devices.

The rest of the paper is organised as follows. Section 2 reviews the related work, while Sections 3 and 4 demonstrate the architecture and implementation of MobileDLSearch respectively. Section 5 illustrates the evaluation of the platform and Section 6 concludes the paper.

II. Related Work

A. Representation of Deep Learning Models

DL models can be different with respect to their layers, associated hyperparameters, features, data and metadata, but these details can be captured and represented in a standard and unified manner across different models. There have been a few studies that use common formats to represent DL models. The study introduced in [8] aims to assist developers with developing and training DL models and uses a JSON format to export the DL models. The DL toolbox proposed in [9] facilitates training DL models by defining complex models using a simple declarative language and uses the YAML format to parse model definitions. A different study presented in [10] uses the Open Neural Network Exchange (ONNX) file format to address the interoperability problem in updating, composing and reusing knowledge in knowledge-centric networking. While the above-mentioned studies show the potential of using a standard format to represent DL models, their focus is mainly on exporting models and addressing application-specific problems.

There are also some attempts to use ontologies to represent DL models. For example, authors in [11] combined DL with an ontology to improve human behaviour prediction. The model was able to learn the individual representations in health social networks from ontologies. In [12], an ontology was used to guide the architecture design of Deep Restricted Boltzmann Machines (DRBM) to improve training and validation of DL models. In a different study [13], authors used a thyroid ontology that was extended with multimedia metadata of images using DL to facilitate identifying and searching through images. Ontologies provide a robust and solid way for representing domain concepts in a common and unified way. Yet, these works use ontologies for representing the application domain concepts instead of the DL models themselves. This makes it hard for users to find an appropriate DL model from a wide range of existing DL models trained on different data sets, developed by different libraries and used for different purposes. This paper proposes a DL ontology and integrates it into a mobile platform that allows users to share and query different DL models through its interactive user interface.

B. Mobile Deep Learning Applications

There has been a significant breakthrough in mobile DL applications since 2010. On-device DL becomes accessible when mobile devices get multi-core processors and powerful graphics processing units (GPUs). A survey in 2018 found a 27% increase in the number of DL-based applications in the official Google Play AppStore within a 3-month investigation period [3]. Nowadays, DL models are nearly ubiquitous among mobile applications and perform a wide range of tasks. These include computer vision (e.g., image classification [14], face detection and recognition [15], augmented reality [16]), natural language processing (e.g., language translation [17], sentence sentiment analysis [18], voice assistant [19], interactive chatbots [20]) and time-series tasks (e.g., human activity recognition [21], gesture recognition [22], music tracking and classification [23]). Previously hand-crafted solutions are gradually replaced by DL techniques because they are more efficient and provide auxiliary methods for intelligent data processing [24].

Although DL performs remarkably in many mobile applications, tailoring DL to mobile devices is still complicated for DL engineers. DL models can be developed via different libraries while not all the libraries allow their models to run on mobile devices without any specific modification, conversion or transfer learning. TensorFlow Lite (TFLite) [25] and Android Neural Networks API (NNAPI) [26] have established a milestone for full-fledged mobile DL pipelines,

but it still lacks sufficient support from other libraries such as Keras and PyTorch. According to the No Free Lunch (NFL) theorem [27], there is no universal model that fits all the problems. Therefore, different DL models with various architectures need to be deployed on real-world workloads to benchmark their performances and select the one that works well for a specific problem. It would be more critical for mobile applications because mobile devices are diverse in their computational resources, sensors and environments. This makes it hard for DL engineers to find an appropriate model from a wide range of DL model zoos, even they realise the reuse of pre-trained models can help them quickly deploy certain applications. These problems inspire us to develop a common platform used to store, query, share and reuse DL models developed for various mobile applications.

C. Deep Learning Model Repositories

Recently there have been a few attempts to enable the storage of pre-trained deep learning models that could be explored and reused by researchers. The examples of such repositories are TensorFlow Hub [28], PyTorch Hub [29] and Model Zoo [30]. While these platforms provide useful capabilities for sharing pre-trained models, they have major limitations. TensorFlow and PyTorch Hubs exclusively maintain the DL models which are developed using their own frameworks, while Model Zoo supports the pre-trained DL models from all development frameworks. All of these three well-known DL model repositories have no conversion function that allows users to transform pre-trained large DL models into mobile lite versions that could be tested on mobile devices and so they can be directly benchmarked on the users' devices. Moreover, the search function of existing models does not support semantic queries at different granularity levels, i.e., existing repositories only provide application level of search results without offering access to the sub-layer details of the DL models such as activation function types. To address these gaps, our proposed MobileDLSearch platform not only enables researchers with more enhanced search for the details of the DL models but also allows them to select a model, convert to a compatible format, download and benchmark it on mobile devices.

III. MobileDLSearch Design

MobileDLSearch follows a distributed design pattern. The DL ontology and DL model instances are stored on the server-side components to handle the increasing number of DL models from constant evolution while the DL model search and integration are run on the mobile client to reduce latency and support real-time analysis. Fig. 1 presents the overall architecture of the proposed platform. The server-side components include:

1) *Ontology Manager* is responsible for mapping different DL models developed using various DL libraries (e.g., TensorFlow, Keras and PyTorch) onto the proposed DL Ontology (presented in details later). This ontology is designed to represent DL models (including their metadata) and the related data set description in a uniform structure.
2) *Model Datastore* is a file system that stores the DL models themselves (i.e., DL model saved files). Each DL model has a unique identifier.
3) *Semantic Datastore Manager* provides persistence capability to store the DL ontology and instances representing DL models (i.e., DL model metadata). It also provides an interface to the Semantic Query Engine.
4) *Semantic Query Engine* provides a SPARQL Processor that allows querying the DL models stored to be retrieved at different levels of granularity.
5) *DL Model Converter Engine* transforms mobile incompatible DL models into the mobile compatible versions.
6) *Web Service* is responsible for interfacing between the mobile client and server-side components. It contains two main APIs: Mobile DL Query API and Mobile DL Download API.

MobileDLSearch consists of five sub-components:

1) *API Manager* serves as the middleware between the user interface and Web Service. It finds/downloads the models from the Semantic/Model DataStore and renders the results back to the user interface.
2) *Mobile DL Model Search User Interface* is responsible for talking to API Manager to search for available models based on users' inputs.
3) *Result View Generator* is responsible for generating the overview/detail model information from the API Manager and representing them in the Mobile DL Model Search User Interface.
4) *Mobile DL Model Integration User Interface* is responsible for using the user-selected DL model to conduct an on-device DL model performance evaluation. It also connects the API Manager to download the corresponding model.
5) *Mobile DL Model Integration* sends the request via API Manger to find the corresponding DL model identifier from the Semantic DataStore and then it goes to the Model Datastore to find the corresponding model. If the model is not available on mobile devices, the search engine will send it to the DL Model Convert Engine and get it back with a mobile compatible version. The Model Datastore will save the converted model in the same location synchronously. The DL Model Compiler is a critical part of the Mobile DL Model Integration. It performs DL-based inference by integrating the Sensor Data with the Mobile Lite DL Model. The Sensor Data is collected from mobile embedded sensors while the Mobile Lite DL Model is downloaded from the API Manager.

Deep Learning (DL) Ontology. Fig. 2 shows the proposed DL ontology key concepts and hierarchies. There are three main concepts in the DL ontology, including *DeepLearningApplication*, *Data* and *DeepLearningModel*. The second level of the ontology includes 11 subclasses, which are divided into further subclasses. The total number of classes considering all the levels is 40. The ontology also encodes the relations

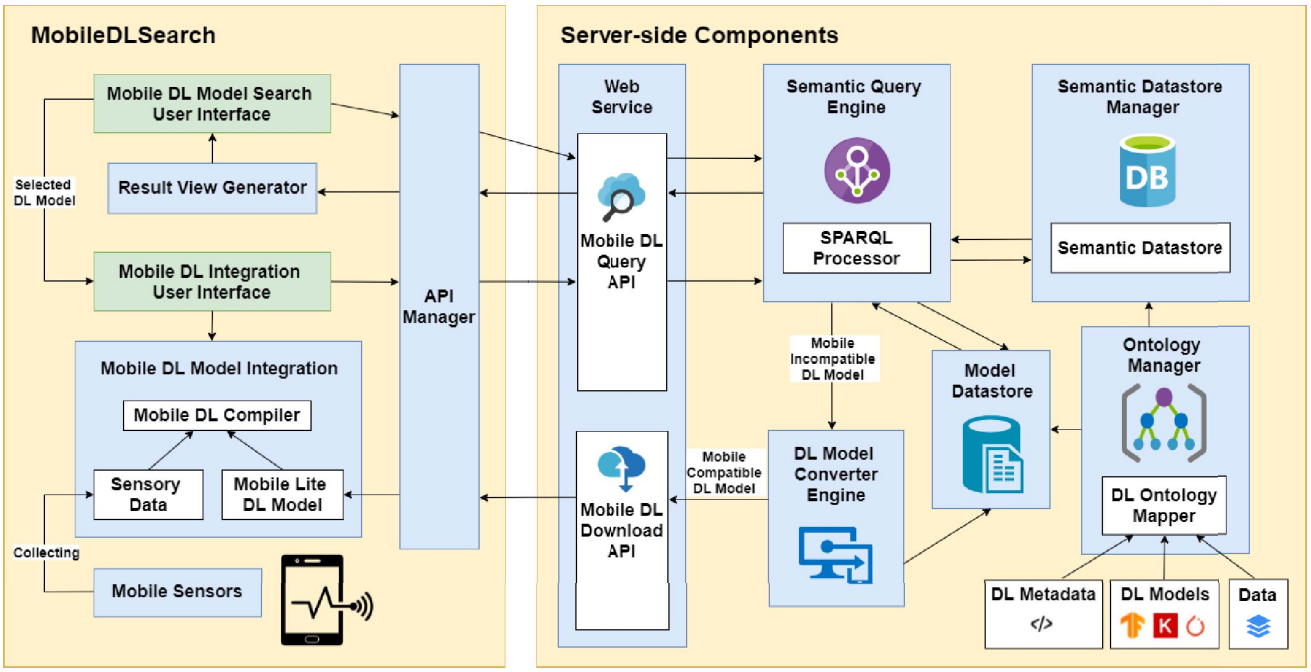


Fig. 1. Architecture of MobileDLSearch platform.

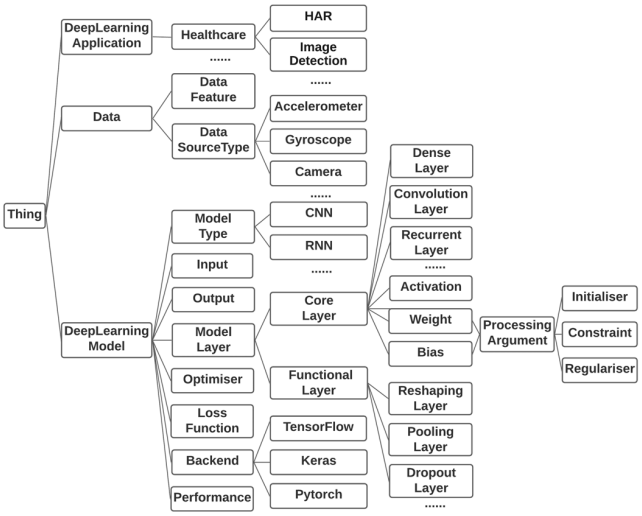


Fig. 2. DL Ontology concepts and hierarchies.

between the classes and subclasses.

DeepLearningApplication describes the applied domain. Due to the limited space, we only present one subclass *Healthcare* under *DeepLearningApplication* and two subclasses *HAR* and *ImageDetection* under *Healthcare*. These subclasses can be extended for other domains (e.g., manufacturing) and other healthcare areas (e.g., clinical natural language processing).

The *Data* concept is a critical part of the DL application. *DataFeature* are the measurable properties of the data and often indicate the output of the DL models. *DataSourceType* refers to where the data is generated and digitised. For mobile application specifically, these include a set of sensors like *Accelerometer*, *Gyroscope* and *Camera*. These subclasses can be extended to other sensors such as magnetometer, pressure sensor, temperature sensor and humidity sensor, etc. In general, *Data* is used to describe data and its characteristics used to develop the corresponding DL models.

The key components of the *DeepLearningModel* are *Input*, *Backend*, *ModelType*, *ModelLayer*, *Output*, *Optimiser*, *LossFunction* and *Performance*. These concepts capture all the important characteristics of a DL model that are common across all models. They also represent the key differences that could be used to compare and distinguish different DL models. For the subclasses under *ModelLayer*, *CoreLayer* includes *ConvolutionLayer*, *RecurrentLayer* and *DenseLayer*. These layers are commonly used to process heterogeneous and temporal data and usually require the hyperparameters such as activation, weight and bias. Therefore, *Activation*, *Bias* and *Weights* are declared as the subclasses under *CoreLayer*. They also have basic *ProcessingArgument* (i.e., *Initialiser*, *Constraint* and *Regulariser*). Other types of layers like *ReshapingLayer*, *PoolingLayer* and *DropoutLayer* are assigned to *FunctionalLayer*.

The ontology also defines the corresponding properties, instances/individuals and their relations according to the "triple" network structure. For object properties, some specific property characteristics are applied to furnish the meaning of the properties. For example, *hasModel* and *isModelOf* are assigned as Inverse Functional Properties, while *hasProviousLayer* that defines the model structure is assigned as Reflexive

Properties because it cannot point to itself. Also, properties like *hasLayer* (model has layer), *hasCoreLayer* (layer has core layer) and *hasConvolutionLayer* (core layer has convolution layer) are assigned as Transitive Properties so that the DL ontology can infer the knowledge such as "Convolution Layer is the subclass of Model Layer". For data properties, they are abstracted based on TensorFlow, Keras and PyTorch. Some similar concepts are integrated (e.g., *ConvolutionLayer*). Both Convolution 1D Layer and Convolution 2D Layer are classified as *ConvolutionLayer* and we use the parameter *ConvolutionRank* to differentiate them. Meanwhile, two approaches are used to address different names that have the same meaning. One approach is to label both of them as synonyms (e.g., *FullyConnectedLayer* and *DenseLayer*). Another approach is only selecting the most widely used and meaningful one.

IV. MobileDLSearch Implementation

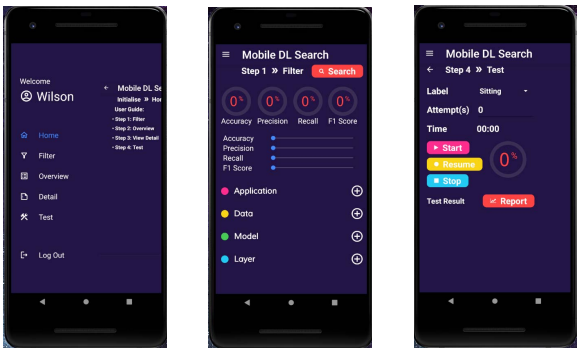
MobileDLSearch is implemented in Java and Python. The mobile client communicates with the server-side components via web service. Table I shows the libraries used in the implementation of MobileDLSearch.

TABLE I
LIBRARIES USED IN THE IMPLEMENTATION OF MOBILEDLSEARCH

Component Description	Language/Library	Version
Ontology Manager	Java/Protégé-OWL	3.4.4
Semantic Datastore	Java/Jena-TDB	3.13.1
Model Datastore	Java/IO	1.8.0
Semantic Query Engine	Java/Jena-ARQ	3.13.1
DL Model Converter Engine	Python/TensorFlow Lite	2.5.0
	Python/Flask	2.0.1
Web Service	Java/Jersey (JAX-RS)	1.19
	Java/Tomcat	8.5.65
Mobile User Interface	Java/Android API	30.0.2
Result View Generator	Java/Android API	30.0.2
API Manager	Java/OkHttp3	3.12.6
Mobile DL Model Integration	Java/Android Sensors API	30.0.2
	Java/TensorFlow Lite	2.5.0

MobileDLSearch has two main functionalities, searching DL models based on users' inputs and integrating selected DL models with sensory data collected from local mobile devices. The demonstration is run on Android Virtual Device (AVD). As shown in Fig. 3(a), Home View is the landing page with a user manual. Filter View, as shown in Fig. 3(b), provides different aspects of filters including application (e.g., application domain, application area, etc.), data (e.g., data source type, data feature, etc.), model (e.g., backend, model type, loss function, optimiser, etc.) and layer (e.g., number of layers, layer order, hyperparameter in each layer, etc.). Users can select one or multiple filters based on their tasks. Fig. 3(c) illustrates the Benchmark View. It allows the user to integrate the selected DL model with mobile sensory data and conduct a real-time on-device experiment to test the model's performance. Considering an example of the DL models for Human Activity Recognition (HAR) tasks, the application allows users to collect data from on-device mobile sensors, deploy DL model inference, compare the results with the ground truth (i.e., the label user selected) and evaluate the real accuracy of the DL model on local devices.

Model converter API is a platform add-on used for converting DL models from desktop/laboratory-based to mobile-compatible versions. The core convert module is developed by the TensorFlow Lite library. TensorFlow Lite is an open-source DL framework developed for on-device inference. It provides a cross-platform solution and can be used in Android, iOS and Linux based embedded devices [25]. The framework can compress a large model to a smaller binary size, which allows users to use hardware acceleration to improve performance for on-device DL inference. The model size can be reduced up to 75% based on different optimisation methods. The framework also provides optimisation tools to adapt DL models on edge devices with limited computational power.



(a) Home View (b) Filter View (c) Benchmark View

Fig. 3. MobileDLSearch user interface demonstration.

V. MobileDLSearch Evaluation

A. Use Case 1: DL Model Search.

Wilson is a musculoskeletal medical researcher who has an IT background. He is currently working on a research project related to knee pain. Due to recall bias, the accuracy of self-reported data by patients about their activities of daily living (ADLs) could be affected. Therefore, he plans to build a mobile application that uses smartphone inbuilt sensors to detect his patients' ADLs automatically. Wilson is looking for a DL model related to the healthcare domain and HAR area. He expects the model to use the data from accelerometer and gyroscope to detect three types of activities: walking, walking upstairs and walking downstairs. Specifically, he wants to find a CNN model (contains convolution layer, pooling layer and dense layer) built by PyTorch, and achieves more than 0.87 in accuracy, 0.88 in precision, 0.87 in recall and 0.86 in F1 score.

By using MobileDLSearch, Wilson is able to enter his specifications and find a DL model that meets his requirements. Since the details of DL models are stored in a hierarchical structure within the DL ontology, they could be queried semantically through SPARQL queries. SPARQL represents each specification in a triple pattern. One examples of DL model

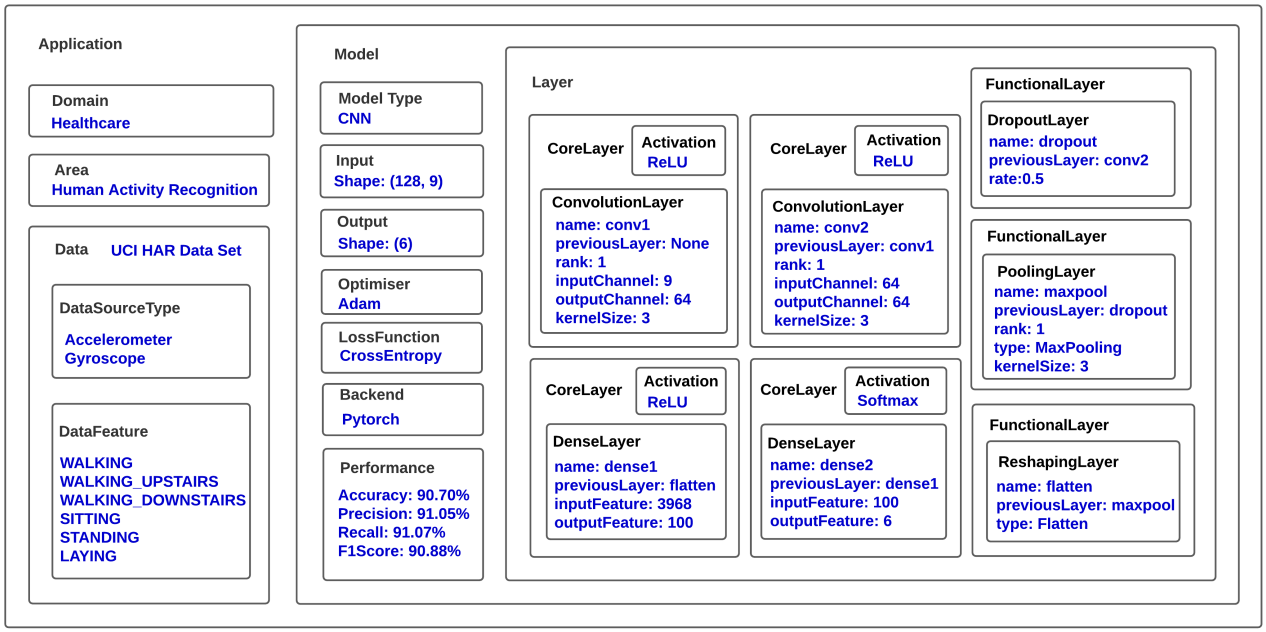
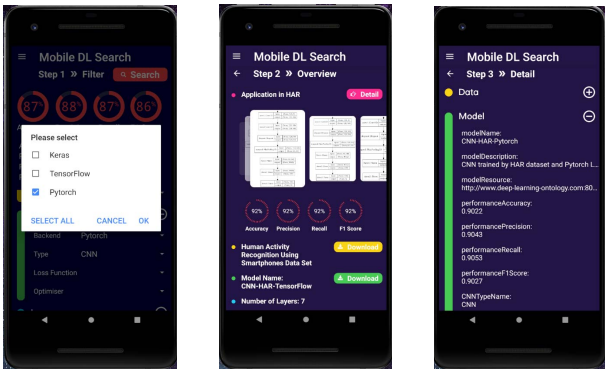


Fig. 4. DL model illustration using the DL Ontology.



(a) Filters (b) Search Results (c) DL Model Detail

Fig. 5. MobileDLSearch search function demonstration.

instances stored in DL ontology and MobileDLSearch demonstration of the search function using in AVD are displayed in Fig. 4 and Fig. 5.

We execute the performance evaluation by querying response time. To examine the impact of data volume, we synthesise 10 original models into 50, 100, 200 and 500 models and stored them in the Semantic Datastore. To examine the impact of query complexity, we employ 3 different SPARQL queries with 15, 20 and 25 query criteria (conditions). The average response time is calculated by electing the mean number from 1000 queries. The results (Table II and Fig. 6) show the performance of the platform is linear, i.e., the response time increases gradually with more number of requests. This indicates the capability of DL ontology and SPARQL to store a large number of DL models and query them from different granularity levels in an effective and efficient way.

TABLE II
RESPONSE TIME WITH THE NUMBER OF MODELS AND QUERY CRITERIA

	15 Query Criteria	20 Query Criteria	25 Query Criteria
10 Models	90 ms	90 ms	90 ms
50 Models	130 ms	137 ms	139 ms
100 Models	167 ms	173 ms	182 ms
200 Models	265 ms	282 ms	290 ms
500 Models	563 ms	586 ms	616 ms

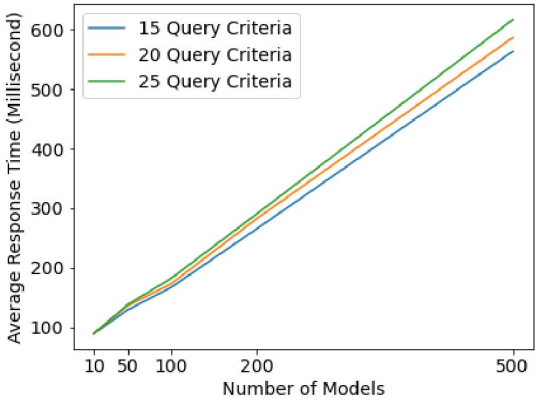


Fig. 6. Response time with the number of models and query criteria.

B. Use Case 2: Obtain Mobile Device Compatible DL Model.

Sally is an anthropologist without much DL background. She plans to use an existing DL model with high accuracy to develop an android mobile application to research human daily activity patterns. However, most of the existing pre-trained DL models she found are targeting desktop computer platforms and cannot be reused directly. These models need to be converted into mobile compatible versions.

To solve this problem for Sally, MobileDLSearch provides a unified mobile DL model converter. The converter is able to convert different desktop/laboratory-based DL models to mobile-based formats (i.e., TensorFlow Lite version). By using this converter, Sally can download the model in a mobile compatible format with one click on the application (i.e., if the model is mobile compatible, it will be downloaded directly; if not, the model will be automatically converted to TFLite format before it is downloaded into mobile devices). Table III and Fig. 7 show the performance test results for converting different DL models. The conversion time is the time consumed for converting DL models while the total time also adds the transmission time consumed to download converted DL models into mobile devices. We conduct the test on a mobile device with 4 central processing units (CPUs) and 16 GB random-access memory (RAM).

TABLE III
CONVERSION TIME AND TOTAL TIME FOR DL MODELS

Model Name	Layer Number	Model Size	Conversion Time	Total Time
VGG16	23	528 MB	47.29 s	52.45 s
MobileNet	92	17 MB	35.69 s	35.70 s
ResNet50	177	99 MB	68.85 s	70.12 s
EfficientNetB0	240	21 MB	78.78 s	79.01 s
InceptionV3	313	92 MB	123.91 s	124.53 s

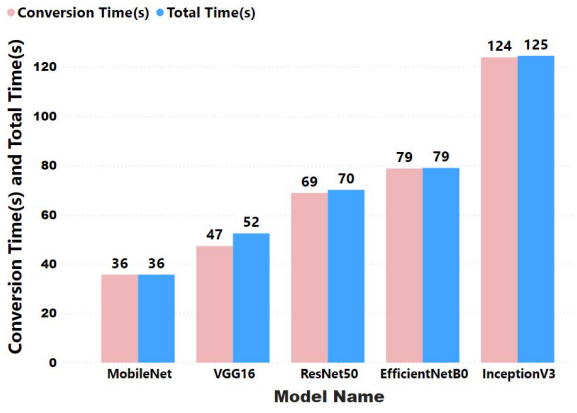
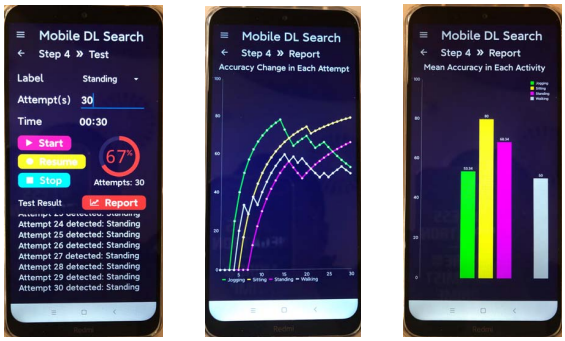


Fig. 7. Conversion time and total time for DL models.

C. Use Case 3: DL Model Integration.

Penny is a DL engineer and pursues to develop a smartphone-based fall detection mobile application for elderly adults. The application is designed to capture abnormal activities from common ADLs (i.e., sitting, jogging, walking, standing) and detect whether it is a fall or not. A DL model needs to be used to effectively recognises these 4 common ADLs. By using the search and download function of the platform, she has already selected one model that meets her requirements and downloaded it into her own device. Now she wants to conduct a real-time analysis by using the smartphone's inbuilt sensors to explore the model performance in each activity and understand how the mobile environments can affect the detection accuracy.

MobileDLSearch provides an easy and interactive DL model integration function within the application. Users can select one or multiple labels as the ground truths and test the real accuracy on their own devices. The application also automatically generates reports for users to review and analyse. The experiment is set in 30 attempts with 4 labels (i.e., sitting, jogging, walking and standing). This means the participant performs each of these 4 physical activities for 30 times and the platform evaluates the detection accuracy. Fig. 8(a) is the user interface of the model benchmarking. The report in Fig. 8(b) and Fig. 8(c) show the DL model achieves higher accuracy in sitting rather than other activities. This result indicates that Penny needs to pay more attention to jogging and walking activities (e.g., collect more data related to jogging and walking to re-train the model) when performing fine-tuning for transfer learning because their accuracies are relatively low. This experiment demonstrates the importance of on-device benchmarking that uses smartphone inbuilt sensors for applications and the usability of the model integration module in MobileDLSearch.



(a) Benchmark (b) Report-a (c) Report-b

Fig. 8. MobileDLSearch DL model integration demonstration.

VI. CONCLUSION

The paper proposes MobileDLSearch, an ontology-based mobile platform for effective sharing and reuse of DL models. The underlying DL ontology presented in this paper provides a standard and generic way to represent different DL models developed using different libraries. The platform enables users with different DL backgrounds to search for DL models of their interests and execute them on their own mobile devices.

The platform also includes the integration of sensory data from mobile devices to benchmark the DL models. The use case-based evaluation of the platform validates its ability to store, query, share and reuse DL models on mobile devices while the system evaluation validates the performance efficiency of the proposed platform.

From a theoretical perspective, this work introduces an ontology to represent, store and query DL models in a unified manner. The ontology provides a standard representation of the DL models (capturing the key differences between these models), the outcomes produced by the models, and the data used by the DL models to produce the outcomes. Metadata of the DL models such as performance metrics (i.e., accuracy, precision, recall and F1 score) are also included in the ontology. Based on the completeness of the ontology, it provides a robust semantic query functionality to retrieve information about the DL models at different granularity.

From a technical perspective, the study makes contributions by developing a platform for sharing the DL models and performing the semantic query. The platform integrates the ontology and the data storage model in the cloud and provides a user-friendly interface for DL engineers to access, share, query, test and reuse DL models in the edge (i.e. mobile device). For experienced DL engineers, the platform provides access to data and the description of the models used to produce the relevant outcomes. This information can then be used to download and reproduce the DL models or reuse the data in other applications where applicable. For other novice users with limited knowledge of DL, the platform provides them with easy access to search and examine the results of data analysis.

While the use cases presented in the platform are targeted at HAR under the healthcare domain, the proposed ontology-based approach also provides a generic way to represent DL models developed for other domains. Future work will focus on extending the platform with compilation capability to allow DL structure to be generated in corresponding DL library format based on the description provided by the DL ontology. We also plan to increase the number of DL models that the platform stores to a large number and conduct performance and scalability evaluation.

REFERENCES

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[2] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu, "Deep learning towards mobile applications," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1385–1393.

[3] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference*, 2019, pp. 2125–2136.

[4] Y. Deng, "Deep learning on mobile devices: a review," in *Mobile Multimedia/Image Processing, Security, and Applications 2019*, vol. 10993. International Society for Optics and Photonics, 2019, p. 109930A.

[5] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, *Python Deep Learning: Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.

[6] Z. Chen, H. Yao, Y. Lou, Y. Cao, Y. Liu, H. Wang, and X. Liu, "An empirical study on deployment faults of deep learning based mobile applications," *arXiv preprint arXiv:2101.04930*, 2021.

[7] K. Nan, S. Liu, J. Du, and H. Liu, "Deep model compression for mobile platforms: A survey," *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 677–693, 2019.

[8] K. M. Lee, K. S. Hwang, K. I. Kim, S. H. Lee, and K. S. Park, "A deep learning model generation method for code reuse and automatic machine learning," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, 2018, pp. 47–52.

[9] P. Molino, Y. Dudin, and S. S. Miryala, "Ludwig: a type-based declarative deep learning toolbox," *arXiv preprint arXiv:1909.07930*, 2019.

[10] D. Sapra and A. D. Pimentel, "Deep learning model reuse and composition in knowledge centric networking," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–11.

[11] N. Phan, D. Dou, H. Wang, D. Kil, and B. Piniewski, "Ontology-based deep learning for human behavior prediction with explanations in health social networks," *Information sciences*, vol. 384, pp. 298–313, 2017.

[12] H. Wang, D. Dou, and D. Lowd, "Ontology-based deep restricted boltzmann machine," in *International Conference on Database and Expert Systems Applications*. Springer, 2016, pp. 431–445.

[13] E. Kim, M. Corte-Real, and Z. Baloch, "A deep semantic mobile application for thyroid cytopathology," in *Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations*, vol. 9789. International Society for Optics and Photonics, 2016, p. 97890A.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[15] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[16] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets deep learning for car instance segmentation in urban scenes," in *British machine vision conference*, vol. 1, 2017, p. 2.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[18] A. Severyn and A. Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, 2015, pp. 959–962.

[19] C. Emmett, D. Dahl, and R. Mandelbaum, "Voice activated virtual assistant," Jan. 31 2013, uS Patent App. 13/555,232.

[20] I. V. Serban, C. Sankar, M. Germain, S. Zhang, Z. Lin, S. Subramanian, T. Kim, M. Pieper, S. Chandar, N. R. Ke *et al.*, "A deep reinforcement learning chatbot," *arXiv preprint arXiv:1709.02349*, 2017.

[21] A. Ignatov, "Real-time human activity recognition from accelerometer data using convolutional neural networks," *Applied Soft Computing*, vol. 62, pp. 915–922, 2018.

[22] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[23] S. Sigtia and S. Dixon, "Improved music feature learning with deep neural networks," in *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2014, pp. 6959–6963.

[24] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "Ai benchmark: All about deep learning on smartphones in 2019," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019, pp. 3617–3635.

[25] "Tensorflow." [Online]. Available: https://www.tensorflow.org/

[26] "Neural networks api." [Online]. Available: https://developer.android.com/ndk/guides/neuralnetworks/

[27] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[28] "Tensorflow hub." [Online]. Available: https://tfhub.dev/

[29] "Pytorch hub." [Online]. Available: https://pytorch.org/hub/

[30] "Model zoo." [Online]. Available: https://modelzoo.co/