AI Folk – Resource Management for Distributed AI

Final Scientific Report
https://aifolk.upb.ro

2022-2024

Contents

1 Introduction **4**

2 AI Folk Ontology **5**

3 AI Folk Interaction Protocol **6**
3.1 Names and Identifiers . . . 7
3.2 Protocol Specification . . . 8

4 AI Folk Deployment **9**
4.1 Entities . . . 9
4.2 Evaluation of Machine Learning Models . . . 10

5 The AI Folk Methodology **11**

6 Autonomous Driving Scenario **14**
6.1 Datasets and Models . . . 14
6.2 AI Folk Ontology Instantiation . . . 14
6.3 Experiments and Results . . . 15
6.4 Enhanced experiment design . . . 17

7 Disaster Response Scenario **18**
7.1 Datasets . . . 18
7.2 Models . . . 19
7.3 Individual Results . . . 20

8 Conclusion **21**
8.1 Project Achievements . . . 21
8.2 Project Impact . . . 22
8.3 Future Work . . . 22

A Competency questions for ML models **25**

B Elements of the AI Folk Ontology **26**

C Python server available operations **28**

Executive summary

Objectives

The main research goal of the AI Folk project was to develop a knowledge model and an interaction protocol which allow a system formed of multiple actors – human, software, or organizational – to find, transfer and use machine learning resources, as well as share improvements on machine learning resources.

The objectives of the project were to:

- O1. develop a semantic description for the resources which can be exchanged in a distributed AI ecosystem, such as individual observations, datasets, and machine learning models.
- O2. develop a protocol that allows actors in an AI ecosystem to search for resources, to understand how resources can be used, and to share resources that they have improved as a result of their own experience.
- O3. define a methodology which allows the integration of existing datasets and models for other applications into an AI Folk ecosystem.
- O4. validate the AI Folk approach through the implementation of prototype applications in two scenarios: an autonomous driving scenario, and a disaster response scenario

All of these objectives have been achieved.

Results

At the end of the project, the following results have been obtained:

- a domain ontology that describes the properties of and relations between four types of resources: individual sensor observations from devices; datasets formed of multiple observations within a specific context (e.g. images of roads in a given area); learning models, together with information about their input, output, and training data; and observations regarding the experience of using a model. Three ontology parts were created:
  - the *core* AI Folk ontology, containing general concepts that can be applied in any scenario;
  - the autonomous driving ontology, containing an instantiation of concepts in the core ontology, as applied to road segmentation models;
  - the disaster response ontology, containing an instantiation of concepts in the core ontology, as applied to flooded area segmentation models;
- a high-level interaction protocol that allows agents in a multi-agent system to communicate with regard to prediction models and share such models.
- a methodology, complete with a deployment infrastructure, specifying how to integrate *core* AI Folk components and how to develop scenario-specific components in order to obtain a complete AI Folk instantiation.
- validation through proof-of-concept scenarios in the two application areas.

It has been predicted at the beginning of the project that a methodology will be created to create SPARQL queries in order to search for prediction models appropriate to a given situation. We have since concluded that using SPARQL queries is both more difficult when instantiating the AI Folk methodology in a scenario, and it also reduces the range of parameters which the queries can rely on. Therefore, we have decided to model queries in a scenario-specific manner, adjusting the AI Folk methodology accordingly.

Impact

The project is expected to have impact on the community of researchers at the intersection of machine learning and multi-agent systems. It is particularly of interest to researchers and industry experts that, rather than using large models which deal with a great variety of data and situations, use smaller, more specialized models, which handle specific situations and narrow distributions of data.

The resources created during the project are available on the project website – https://aifolk.upb.ro. In terms of publications, there have been three publications disseminating results of this project.

The conference paper presented at the EMAS 2023 Workshop at AAMAS 2023 presented the underlying model of the infrastructure used for the AI Folk deployment, emphasizing on the formal foundation allowing the FLASH-MAS multi-agent framework to work with a variety of resources, including machine learning prediction models. The paper generated very interesting questions and lead to discussions with leading researchers in the EMAS community.

The journal paper in Sensors, also in 2023, presents a distributed communication protocol – the Websocket Regions protocol – which can be used in decentralized, distributed deployments such as the ones envisaged by the AI Folk project, which has been validated using the FLASH-MAS framework, and which serves as an underlying layer for the AI Folk interaction protocol.

The project was also disseminated at the Management Committee Meeting of COST Action CA19134 – "Distributed Knowledge Graphs" in September 2023 and the final results will be disseminated at the 2024 management meeting of the action.

The conference paper presented at DCAI 2024 presents the AI Folk approach and methodology, based mainly on the autonomous driving scenario. It garnered relevant questions regarding the infrastructure and the methodology.

A project proposal – WAIz: Hypermedia-based agents using foundational models and knowledge exchange to create personalized plans – partially based on the results of the AI Folk project, meaning to apply the AI Folk methodology in a more concrete scenario, intends to use the same approach, but focused on large language models applied for planning in a smart building scenario.

1 Introduction

As machine learning (ML) continues to permeate every aspect of daily life, gathering datasets and training state-of-the-art models becomes more and more the privilege of large technology companies. Approaches such as *federated learning* [Li et al., 2020] and *edge AI* [Wang et al., 2019] propose the decentralization of learning, such that model training happens on devices which are closer to the user, using that which resides on those devices. One of the core challenges of federated learning is heterogeneity of devices and openness of the distributed AI system [Li et al., 2020].

Moreover, withe the advent of Large Language Models (LLMs), two different approaches have become evident: one using larger LLMs, very costly to train, handling very many context and usages, and one using smaller, more specialized LLMs, as weak learners [Shen et al., 2024]. In the latter case, the challenge that arises is choosing which model is more appropriate for each context.

Our research goal was to develop a comprehensive solution, comprised of a knowledge model, an interaction protocol, a deployment infrastructure, and an instantiation methodology, that allows a system formed of multiple actors – human, software, or organizational – to find, use and share improvements on machine learning resources. Such resources can be datasets, models, or experiences.

This goal involves several challenges, some of which are research-related and some of which are practical, but all deal mainly with the possibility of finding parts of the solution which are scenario-invariant, which can be applied, without change, to any application domain.

An important challenge was therefore the modelling of an AI Folk deployment in general, involving the decision of which components are generic and which are scenario-specific, and which need to be split into parts. For instance, the ontology for any concrete deployment is made of two segments¿ the AI Folk core ontology, which is generic, and a scenario-specific instantiation of the core concepts that can be used at runtime to semantically represent knowledge about the models used by agents in the scenario.

An important practical challenge was the fact that most machine learning (ML) models are implemented using Python libraries, whereas the deployment framework, as well as the libraries for working with ontologies, are implemented in Java. That meant developing a mechanism for interaction between the two languages, which is based on a local Python RESTful web server which loads and manages the ML models, and what we call a *driver* which assists the components implemented in Java with accessing the server.

A challenge also related to the ML models was that there is a great deal of variety in how ML models are evaluated (executed), how input must be transformed in preparation for the model, and how the output of the model must be processed after the model evaluates. This variety was so great that the solution was to implement specific pieces of code for each of the models integrated in our implementation, to deal with each one of these steps – input transformation, model evaluation, and output processing.

In order to have correct information about a model, one needs to have information about the dataset on which the model was trained, or, in case this is not available, one needs to evaluate the model on an appropriate dataset. On the dataset, information relevant to the scenario must be gathered. For instance, for the autonomous driving scenario, for dataset we needed to obtain information such as weather conditions, number of cars and pedestrians, and so on; and we needed to evaluate models on the datasets in order to assess their performance.

The chapters that follow give details on the results of the project: the AI Folk ontology, the AI Folk interaction protocol, the deployment infrastructure, the AI Folk methodology, and the two proof-of-concept implementations.

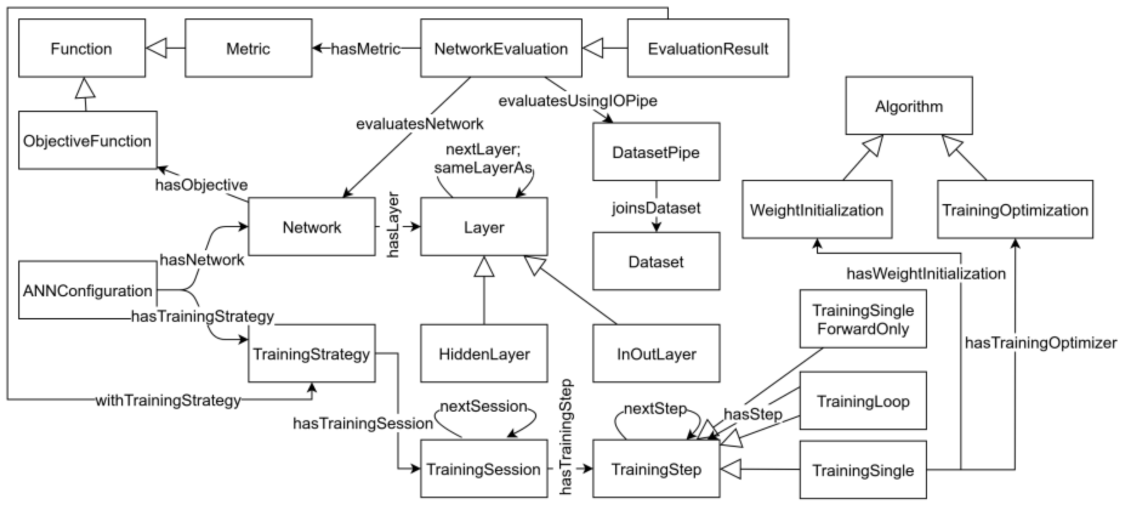


Figure 1: The ANNETT-O ontology architecture proposed by [Klampanos et al., 2019]

2 AI Folk Ontology

In order to develop the AI Folk ontology, we have tried to understand which are the specific challenges in deployments such as the ones envisioned by the AI Folk project, as well as in autonomous driving scenarios and most specifically semantic image segmentation tasks, which are among the most challenging in such scenarios. We have hence developed a list of *competency questions* which the description of a machine learning model *is supposed to answer*. The list of questions is available in Appendix A.

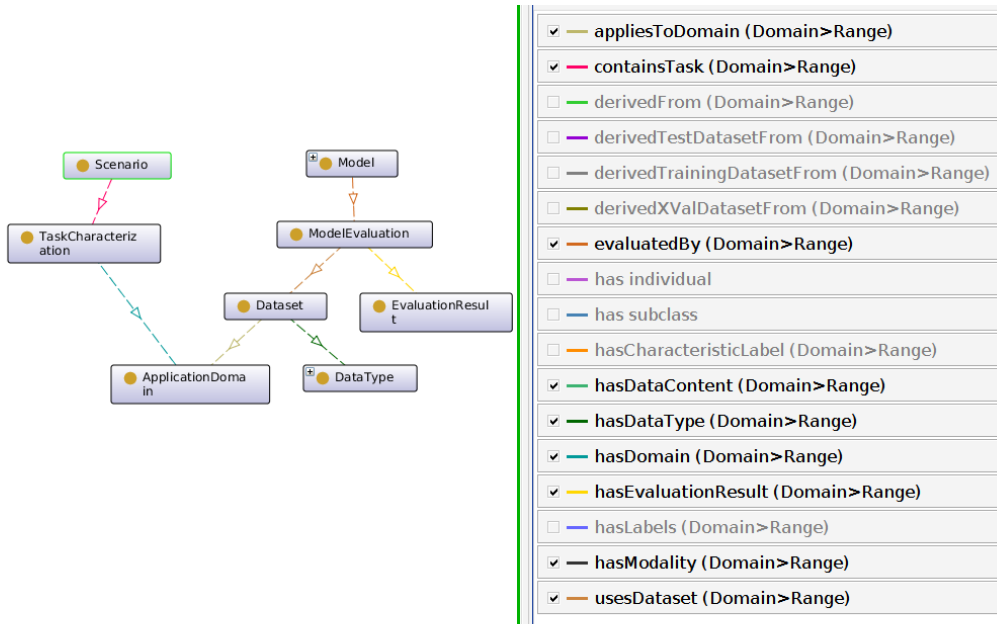
In AI Folk, the ML models we focus on are based on deep neural networks (DNN). The vast architectural space of DNNs makes the selection of an appropriate architecture for a task challenging. Furthermore, it enables the development of knowledge bases that support efficient querying, analysis, and comparison of various neural network architectures and experiments. Ontologies emerge as a promising solution to describe DNNs in a structured manner, enabling support for querying, analysis and comparisons of various DNN architectures, as well as their configuration, training methodologies and evaluation procedures.

FAIRnets [Nguyen and Weller, 2019], for example, is an RDF dataset that stores information about existing neural networks. Using RDF and OWL, the system enables SPARQL querying for desired characteristics of DNNs. Users receive search results comprising neural networks meeting the query criteria, facilitating insights into existing architectures. Additionally, detailed information and further links about network architectures are provided.

In AI Folk we choose to extend the ANNETT-O ontology [Klampanos et al., 2019] (see Figure 1), which offers a structured and computationally accessible vocabulary, enabling systematic documentation and analysis of DNN design and experimentation, with emphasis on aspects of evaluation, topology, and training.

The purpose of the AI Folk ontology is to introduce the *vocabulary* required to exchange such messages: (i) describe the data in terms of its *qualities* – i.e. provide a **data context description**, and (ii) describe the AI models and their *evaluation* mode, so as to determine if they are suitable for a given *data context*.

Describing data in terms of its *qualities* is by necessity an *application domain* dependent procedure. Consequently, the AI Folk ontology is designed in a modular fashion. It contains a **core** vocabulary



- ✓ appliesToDomain (Domain>Range)
- ✓ containsTask (Domain>Range)
- ☐ derivedFrom (Domain>Range)
- ☐ derivedTestDatasetFrom (Domain>Range)
- ☐ derivedTrainingDatasetFrom (Domain>Range)
- ☐ derivedXValDatasetFrom (Domain>Range)
- ✓ evaluatedBy (Domain>Range)
- ☐ has individual
- ☐ has subclass
- ☐ hasCharacteristicLabel (Domain>Range)
- ✓ hasDataContent (Domain>Range)
- ✓ hasDataType (Domain>Range)
- ✓ hasDomain (Domain>Range)
- ✓ hasEvaluationResult (Domain>Range)
- ☐ hasLabels (Domain>Range)
- ✓ hasModality (Domain>Range)
- ✓ usesDataset (Domain>Range)

Figure 2: An overview of the main concepts and properties in the AI Folk Core ontology module, showing how Scenarios described in terms of Tasks relate over an Application Domain with ML Models and their Evaluation on Datasets from the same domain

defining how concepts such as a *Scenario* and its *TaskCharacterization* are related to a *Model* and its *ModelEvaluation* in an *ApplicationDomain* (see also Figure 2). The core module is then extended by *domain specific* vocabulary (such as the segmentation in autonomous driving one) which introduces concepts and relations that describe *qualities* of data from key perspectives of the application domain (see Section 6.2)

Figure 2 gives a general overview of the AI Folk core concepts and relations. The left side of the concept graph starts from a *Scenario* that is described in terms of *TaskCharacterizations*. Tasks are subdivided from the perspective of a ML algorithm into objectives such as *Classification*, *Regression*, *Segmentation*, *ObjectDetection*. Each such task is *has a domain* to which it applies. The *ApplicationDomain* is also one that was addressed by *Dataset*. The *Dataset* in turn was used during the *evaluation* of a *Model*, obtaining an *EvaluationResult*. In this way, scenario tasks (as perceived by the agent) are linked to models that have been on datasets that have similar content.

Both *TaskCharacterization* and a *Dataset* are further described by one or more *DataContexts*, which is the concept that gets extended in the domain specific ontology module.

In Section 6.2 we detail how the AI Folk *DrivingSemanticSegmentation* ontology module addresses these modeling issues and Section 7 details the instantiation of the ontology on the Disaster Reponse scenario.

3 AI Folk Interaction Protocol

The purpose of the AI Folk interaction protocol is to ensure that agents can search for models which are appropriate to a situation and can transfer information about those models. While initially we intended that search messages contain a SPARQL query that can be applied directly by the local Ontology Driver in order to identify appropriate models, while developing the experiments it resulted that sometimes the query would have to be overly complex to create, given the constraints of working with ontologies. Conversely, it resulted, while implementing situation detection, that situation detection itself required using some machine learning modes (e.g. for determining the weather, or the average number of pedestrians in a given time window), hence there is a finite and determined number of features of the situation that one can handle in a scenario. Given this determination, it resulted that instead of creating a SPARQL query and let the Ontology driver run it, it is easier, from the point of view of the developer implementing the scenario, to iterate through model descriptions and check for the appropriate features.

In conclusion, when an agent needs to search for a model with certain properties, it assembles an RDF graph describing a model with the needed features. It then serializes this description and sends it, in a message, to other agents.

3.1 Names and Identifiers

In any multi-entity system [Olaru et al., 2023], there is an important need for being able to identify things. In AI Folk there are two elements that need to be identifiable across the system: entities and models. At this point we assume datasets as not changing and, moreover, we consider they are not stored inside the AI Folk system. Hence, their identification is trivial and can be performed, for instance, by means of their URI.

Entities need identifiers so that they can be addressed as destinations of messages. Identification must work in the context of entities which are bound to mobile devices (cars, mobile robots, smartphones), and in the context of a changing network topology. Considering our work in developing a decentralized communication infrastructure [Olaru and Pricope, 2023], it would be easiest to identify entities using their home server URI, concatenated with an identifier specific to the entity and unique to the home server. The home server paradigm [Wojciechowski, 2001] enables a distributed approach, while managing parts of the system in a centralized manner – all entities are assigned to a home server which knows where the entity is located. If a device is routable (from an Internet Protocol point of view), then it can serve as a home server for all entities that execute on that device.

Machine learning models also need to be identified. In a deployment in which entities search for, exchange and share ML models, entities must be able to refer to those models. An additional challenge comes from the fact that models can be tweaked by individual entities, leading to slight variations in the model. We propose the following identification mechanism.

Every model is identified by a *hash number*, providing a unique identifier. Moreover, in order to be able to differentiate between models which are completely different and models which are only slight variations of other models, we build a conceptual *model forest* of hashes, as follows:

- there is one *tree* of hashes in the forest for each major model existing in the system (including its variations and tweaks);
- there is a *node* in the tree of a model for each concrete implementation of the model. In the tree structure, the node only contains the hash of the model. For a node corresponding to a specific model, the children of that node correspond to models which are directly derived from that model. Edges optionally contain summary information about the variation which has been applied.

Of course, in a large system, there will be no single entity which holds the *entire* model forest of hashes. However, individual entities using ML models will find useful to store, for each of the models they use, the ancestor line and potentially other parts of the hash tree. Model libraries will also store complete or almost-complete trees for the models they store.

Message type	Message arguments		
DISCOVERY segment			
SEARCH	Query	Area	
EXPERIENCE SEARCH	#modelHash	Area	Subtree distance
LISTING	Query id / #modelHash		
TRANSFER segment			
REQUEST transfer	#modelHash	[chunk size]	[chunk no]
TRANSFER	#modelHash	chunk size	chunk no
UPDATE segment			
SUBSCRIBE	#modelHash	Subtree distance	
SUBSCRIBE	Query		
UNSUBSCRIBE	#modelHash / Query id		
MODEL UPDATE	#modelHash	#newModelHash	update description
REQUEST diff	#modelHash	#newModelHash	
TRANSFER diff	#modelHash	#newModelHash	

Figure 3: Types of interactions in the AI Folk protocol.

3.2 Protocol Specification

In order to allow entities in an AI Folk ecosystem to discover and exchange ML-related resources, including ML models, we developed the protocol presented in this section. The protocol contains three segments: discovery, transfer, and update. The protocol is high-level and does not cover issues such as message routing or conversation control.

Discovery is the process through which an entity can search for (1) ML models adequate for the situation and needs of the entity and (2) information on the experiences other entities had with using specific models in specific situation.

An entity will be able to search for a model according to a given query, which is a serialization of an RDF graph, and in a given area around its location (in terms of physical space or in term of in terms of network space). An entity will also be able to query the experience with a particular ML model; the variables are the area to search (in physical or network space), but also the maximum *subtree distance* in the tree of models – that is, the entity may search for experience data not only with a specific model, but also with closely related models. Here, by experience we signify performance metrics related to the use of models in a particular situation. The destination of the search will reply with a listing of results.

Transfer is the process through which entities can transfer among each other information about ML models. Information about a model comprises three main parts:

- the semantic description of the model, as an RDF graph, which can be integrated in the agent's local graph;
- additional code (in Python) required to run the model, to process its input and its output;
- the model itself (e.g. as a `.pth` file containing structure data and weights) or an address from which the model can be downloaded to the local node.

Update is the process through which entities may subscribe for and receive updates on models of interest or queries of interest. Since some changes to models may be applied incrementally, an entity is able to request information on how to update its own model in order to obtain another version, without the need to transfer the entire new version.

4 AI Folk Deployment

During the AI Folk project, we had an important focus on the practical aspect, on the possibility to actually deploy an AI Folk scenario and deal with all the modelling and technical issues that arise.

4.1 Entities

In order to create an AI Folk development test-bed and to run experiments with the two scenarios, there was a need to create or use an infrastructure that:

- supported the deployment of various entities and the management of their lifecycle. Among these entities there are agents, ML models, communication infrastructures, and auxiliary elements such as running the scenario or working with ontologies and ML models;
- enabled communication among the various entities, especially between agents;
- enabled rapid testing of multi-agent scenarios.

In order to satisfy these requirements without creating a development test-bed from scratch, we have decided to use the Flash-mas framework – A Fast, Lightweight Agent Shell Multi-Agent System – developed at UNSTPB in recent years [Olaru et al., 2019, Olaru et al., 2023]. We have chosen Flash-mas over JADE [Bellifemine et al., 1999], a popular MAS deployment framework, due to its flexibility, support for dynamically loaded components, the ability to create new, custom, first-class entities, and good performance [Olaru and Pricope, 2023].

In short, Flash-mas, implemented in Java, is able to deploy any persistent entity given it implements a basic `Entity` interface. There are a few standard entities in Flash-mas, such as nodes, pylons (embodying communication infrastructures), agents, and shards. *Shards* are sub-agent entities which embody specific agent functionality, such as messaging, or monitoring.

We have used the standard Flash-mas entities for nodes, agents, and communication infrastructures (using the decentralized Websocket Regions protocol [Olaru and Pricope, 2023]). We have used standard agent shards such as messaging and monitoring. In order to deploy the AI Folk experiments, we needed to build several novel entities, as described in the following paragraphs. A class-interaction diagram presenting the relations between these entities is presented in Figure 4.

Several **drivers** had to be created and designed, for interaction between AI Folk agents and their environment. We call *drivers* persistent entities that enable the relation with node-local non-agent entities (different from pylons, which handle the relation with network-wide infrastructures, e.g. for communication). We have created three drivers for each node in the deployment, to deal with: **scenario execution** – feeding input data to agents and checking their output; **ontology** access and querying (see Section 2); and evaluation of ML models – see Section 4.2.

Several novel agent shards had to be implemented as well, embodying the functionality needed to implement the AI Folk methodology. The **Scenario shard** receives input from the Scenario driver and feeds it to the agent's event queue. It also intercepts the agent's actions and sends them to the Scenario driver for verification and monitoring. The evaluation of ML models on the input, in order to decide the action, is done by the **ML Pipeline Shard**, which uses the currently selected ML models.

The actual implementation of the AI Folk methodology, in terms of evaluation and selection of the appropriate models to use, as well as searching for other, better models to use in the current situation, is done by the **ML Management shard**.

In order to correctly model the isolation of scenario-specific code, for instance for the evaluation of the current situation, we have created a novel type of sub-agent entity – the **Scenario Feature**, which encapsulates all the code which is specific to a scenario and which is loaded dynamically based on the

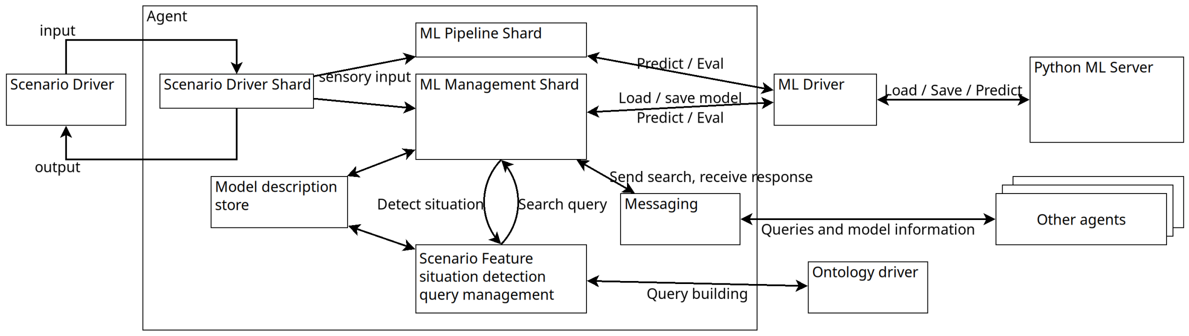


Figure 4: A diagram presenting the relations between these entities in an AI Folk deployment

```
quick.Boot -load_order driver;pylon;agent
  -package testing net.xqhs.flash.ml src-experiments aifolk.core aifolk.onto
  -loader agent:composite
  -node node1
    -driver ML:mldriver
    -driver Scenario:scen script:script
    -driver Ontology:ont
    -pylon WSRegions:Pylon1 isServer:<localhost>:8885
    -agent A in-context-of:ML:mldriver in-context-of:Scenario:scen
      -shard scenario -shard MLPipeline
      -shard MLManagement feature:AutoDrive -shard messaging
```

Figure 5: Deployment configuration for one node in an AI Folk deployment.

deployment configuration.

Thanks to the advantages of Flash-mas , deployment is easily specified via an XML file, or via the command line, in a simple and intuitive manner. For instance, for our test scenario, the command line for one node is shown in Figure 5.

In this specification, the important highlights are the three drivers, and the specification of one agent (all agents are specified similarly), which is placed in the context of the three drivers, so it can access their functionality, and which contains functionality for messaging, for receiving scenario-related events (the Scenario shard), for using ML models (the ML pipeline shard), and for managing its own ML models, complete with the specification of the feature to be used – specific to each type of scenario.

4.2 Evaluation of Machine Learning Models

An important challenge in the deployment of AI Folk scenarios was the fact that the Flash-mas framework, as well as the components for the management of RDF graphs, are implemented in Java, but most machine learning models are implemented in Python, and using dedicated ML libraries such as PyTorch[1] and Tensorflow[2].

Integrating a Python Flask server for neural network predictions into a Java project can offer significant advantages in terms of flexibility, scalability, and ease of deployment. Python, with its rich ecosystem of machine learning libraries like PyTorch and TensorFlow, excels in building and training neural networks. Flask, a lightweight web framework, provides a seamless way to expose these models through RESTful APIs. By employing a Python Flask server, we can encapsulate the complex neural network

[1]PyTorch: pytorch.org
[2]Tensorflow: www.tensorflow.org

```
PORT: 5000
MODELS:
  - name: "MobileNetV2"
    path: "models/mobilenetv2.pth"
    transform: "datasets.transform.ImageNetTransform"
    cuda: false
    input_space: RGB
    task: "classification"
    dataset: "ImageNet"
  - name: "ResNet18"
    path: "models/resnet18.pth"
    cuda: true
    input_space: RGB
    transform: "datasets.transform.ImageNetTransform"
    task: "classification"
    dataset: "Custom Dataset"
DATASETS:
  - name: "Custom Dataset"
    class_names: ['apple','atm card','cat','banana','bangle','battery','bottle','broom',
                  'bulb','calender','camera']
```

Figure 6: Example configuration for prediction models and the dataset they were trained on.

logic in a language optimized for machine learning, while Java, renowned for its robust enterprise solutions, can focus on handling other aspects of the project. This separation of concerns facilitates a more modular and maintainable codebase.

For the server, we chose to use a YAML configuration file where we define the initial models and datasets. For each model, we specify its name, the path to the .pth file containing the model weights, the input type processed by the model, the task for which the model is proposed, the dataset used for training, and information related to any required preprocessing. Similarly, for each dataset, we specify its name and the possible labels – this is because many times models do not also specify the output labels. To better understand the proposed format, we provide an example of a configuration file in Figure 6. When a new model is added, its metadata is added in the same format.

Following the concepts in the entity-operation model, for each service, the route is defined using the `@app.route` decorator with the path (e.g. `/predict`). It specifies that the route can only handle HTTP POST requests (`methods=['POST']`). The services provided by this server, which can be used from the Java components via the `ML Driver`, are the following (detailed in Appendix C):

- adding information about a new prediction model (`add_model`)
- adding information about a dataset (`add_dataset`)
- requesting a prediction from the model, for a given input (`predict`)
- obtaining a list of models (`get_models`)
- exporting information of a model so that it can be ready to transfer to another agent (`export_model`)

5 The AI Folk Methodology

The end goal of our project is to develop a methodology that allows the integration of prediction model searching and transfer into a multi-agent deployment in any scenario. Since the beginning, we knew that in deploying the methodology for a given scenario, some parts of the implementation will remain invariant and some parts will need to be tailored to the necessities of that scenario. Having analyzed the deployment of the AI Folk approach in both scenarios – autonomous driving and disaster response – we have identified invariant and scenario-specific elements as presented in Table 1 and we have constructed the methodology as follows.

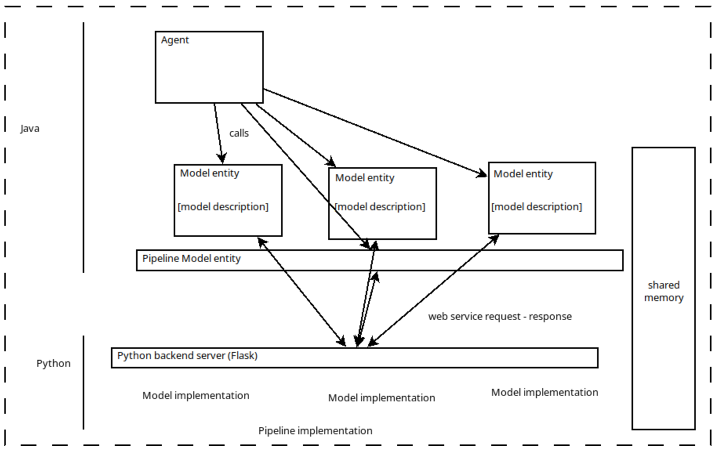


Figure 7: A view on the interaction between Java components and Python components in an AI Folk deployment.

The AI Folk approach applies to agent systems in which participants have to execute their predictive tasks in environments in which the degree of input variability is too large to be covered by a single prediction model. Such situations require agents to form *experience sharing communities*, in which agents collaborate to exchange appropriate prediction models with each other. The mentioned operating conditions necessitate the support of a framework that is based on the following design principles and requirements.

1. Formation of experience sharing communities: refers to giving support to agents to dynamically identify and communicate with the set of peers that have a *relevant experience*, in terms of similarity of task and data descriptions on which a prediction model has been successfully employed (where success is measured based upon standard or mutually agreed upon task-specific metrics). Thus, the agent community building principle is driven foremost by an information retrieval concern, as opposed to one related to logical / physical vicinity or speed of communication.

2. Identification of salient prediction model, task and data characteristics: refers to enabling to agents to describe *what* is changing in the task for which they have to use prediction models. AI Folk enlists the use of semantic web technologies, such as ontologies, to describe task types, general and task-specific data attributes, DNN model architectures, DNN model evaluation datasets and evaluation performance. Note that we currently focus on the case where data characteristics that agents should pay attention to (because they can cause distribution-shifts) are defined based on human domain knowledge. The methodology can be extended in future work to include approaches where a data-driven model itself is responsible to highlight and semantically map the data characteristics which it considers are relevant in the prediction and which have changed.

3. Transfer, loading and running of prediction models: refers to giving support to agents to reference shared models, transfer model parameters, dynamically configure a model deployment environment and run the inferences required by their prediction task. AI Folk envisions a system that supports both (i) cloud-based setups, where agents run predictions in an already configured cloud environment for which a web-based entry point and access credentials are provided when sharing it with other agents, and (ii) an edge-based setup that enables local running of prediction models and which requires methods to efficiently serialize and transfer model parameters and deployment configurations (in a manner similar to Federated Learning setups). Note that in AI Folk we currently focus on building support for cloud-based setups.

Using the AI Folk methodology in a given scenario consists of the following steps:

Table 1: Invariant and scenario-specific elements in deploying the AI Folk methodology.

Component	Invariant	Scenario-specific
AI Folk Ontology	Core concepts related to data contexts, datasets, functions, models, model evaluation, task characterizations, neural networks and optimizers, etc.	Specific concepts. E.g., for the autonomous driving scenario, the driving scene, environment conditions. Instances of specific tasks, datasets and models that can be used in autonomous driving scenario
AI Folk Interaction Protocol	The protocol and its integration with the FLASH-MAS framework are invariant.	A difference appears in the case in which models are not transferred, but obtained through other means (e.g. through functions in the libraries). In this case the `REQUEST` and `TRANSFER` primitives are no longer necessary.
Queries	Queries are serialized and sedn between agents in a scenario-independent manner.	Queries are built and evaluated in the `Feature` entity depending on how the parameters in each scenario are evaluated and compared.
FLASH-MAS entities for AI Folk	Developed entities are scenario-invariant.	Situation recognition code is scenario-specific and integrated in the `Feature` scenario-specific entity.
ML Python server	The implementation of the server is invariant, but the libraries that must be installed on the host machine and which the server imports are scenario-specific.	The libraries and the drivers of each individual model are scenario specific.
Code for situation recognition		The code is specific to model and to the given scenario, as they evaluate scenario-specific parameters, and is integrated in the `Feature` entity.

1. for each prediction model, create instances in the ontology, describing the model and the datasets on which it was trained; if necessary, additional concepts and relations can be added in the methodology;
2. implement code that evaluates specific parameters on input data and on used datasets, such that it can be used to identify if the current situation – the current distribution of input data – sufficiently matches the distribution in the dataset on which the model was trained on;
3. implement code for input transformation, evaluation, output processing, and library import, for each model in the scenario;
4. decide on the appropriate manner of transferring models between agents, depending on model size and network capabilities;
5. integrate in agents decision code for selection of the models, depending on scenario-specific parameters for models;
6. deploy agents according to the scenario requirements.

While the list of items to cover may seem long, we posit that applying the AI Folk approach and using the AI Folk infrastructure greatly reduces the time taken to deploy a MAS which is able to exchange information on prediction models, since all the scenario-invariant code and structures are already implemented and appropriately modularized. What is left to do are, in fact, steps that would be necessary anyway.

The code for the various AI Folk experiments is open-source, currently in the FLASH-MAS Github repository[3], in the various `aifolk` branches.

In the following, we will show how the AI Folk methodology was applied to two different scenarios – one in the field of autonomous driving and one in the field of disaster response.

[3]FLASH-MAS Github repository: https://github.com/andreiolaru-ro/FLASH-MAS

Table 2: Semantic segmentation datasets

Name	ADE20k	Cityscapes	Pascal VOC	Stanford	CamVid
created	2016	2016	2012	2017	2008
domain	general	driving	general	indoor	driving
tasks	object detection, semantic segmentation	detection, segmentation	classification, detection, segmentation	detection, segmentation, depth	object detection, semantic segmentation
no. classes	150	30	20	24	32
illumination	generally day	daytime	generally day	generally day	daytime
weather	generally sunny	good/ medium weather	generally sunny	indoor/ artificial light	mostly generally sunny

Name	KITTI	BDD100k	Vistas	SUN	Synthia
created	2013	2020	2017	2017	2016
domain	driving	driving	driving	mostly indoor	virtual city
tasks	multiple tasks (detection segmentation etc)	detection, segmentation	detection, segmentation	detection, segmentation, depth	semantic segmentation, detection
no. classes	11	19	124	37	13
illumination	daytime	day, dusk, night	various day time	generally artificial light/ day	daytime
weather	generally sunny	sunny, rain, overcast, snow	various day times	not applicable	sunny, overcast

6 Autonomous Driving Scenario

The objective of implementing a proof-of-concept scenario was to validate the steps needed to apply the AI Folk methodology and to verify the mechanism through which prediction models are validated, queried, exchanged, and switched.

6.1 Datasets and Models

For the autonomous driving scenario, we have showcased several datasets and models related to the task of road segmentation in RGB images:

- Datasets: ADE20k [Zhou et al., 2017], Cityscapes [Cordts et al., 2016], Pascal VOC [Everingham et al., 2010], Stanford [Armeni et al., 2017], CamVid [Brostow et al., 2009], KITTI [Geiger et al., 2013], BDD100k [Yu et al., 2020], Vistas [Neuhold et al., 2017] as well as Sun RGB-D [Xiao et al., 2010] and Synthia [Ros et al., 2016];
- Models: DeepLabv3 [Yurtkulu et al., 2019] and YoloV8 [Jocher et al., 2023], which are state-of-the art segmentation models.

The most important information regarding the datasets can be found in table 2. Besides the most relevant properties described in the tables, each dataset was evaluated according to a number of questions regarding the average number of cars or pedestrians per image, average percentage for the cars or the pedestrians in an image, the percentage of the weather conditions and the illumination conditions and also considering the number of different intersections in an image. These questions could be further used in order to evaluate the benefit of using one dataset against another.

6.2 AI Folk Ontology Instantiation

Remember from Section 2 that the AI Folk ontology is developed in a modular way, with the core vocabulary being extended by a domain specific one. In the driving scene segmentation scenario for autonomous driving a vocabulary to describe a *DataContext* specific to such an application domain is

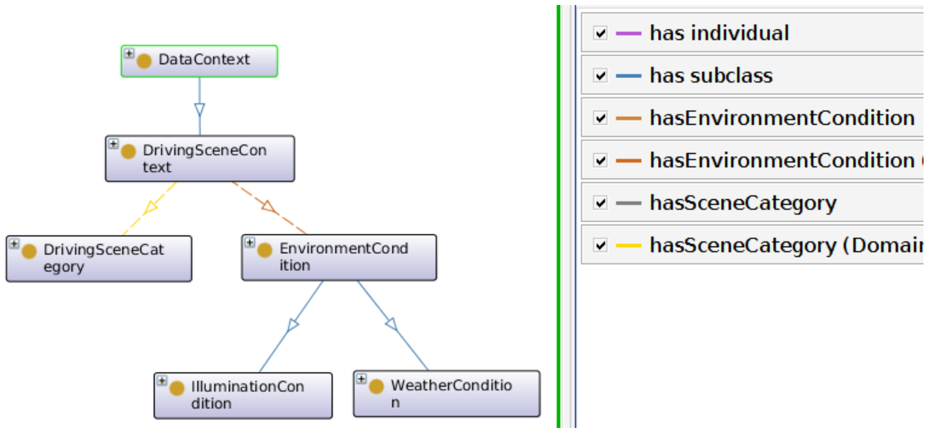


Figure 8: The concepts that characterize a *DrivingSceneContext* in the Segmentation for Autonomous Driving application domain

defined.

Figure 8 shows how the *DataContext* concept from the *core* AI Folk ontology is extended by a *DrivingSceneContext* specific one. The driving scene context is characterized by an *environment condition* (which can refer to the illumination condition or to the weather condition) and a *driving scene category* (e.g. a city, rural, highway or parking area).

Furthermore, the *competency question* based ontology development methodology explained in Section 2, gives rise to *instances* of the *DrivingSceneContext* which have *DataProperties* as exemplified in Figure 9.

In Figure 9 it is important to note the extensive set of *data property assertions* which were identified through the *competency question* methodology and which are deemed likely to influence the performance of a segmentation model. Specifically, the data properties describe of some key situations and objects that are likely to be encountered in an autonomous driving scenario. They relate to aspects such as:

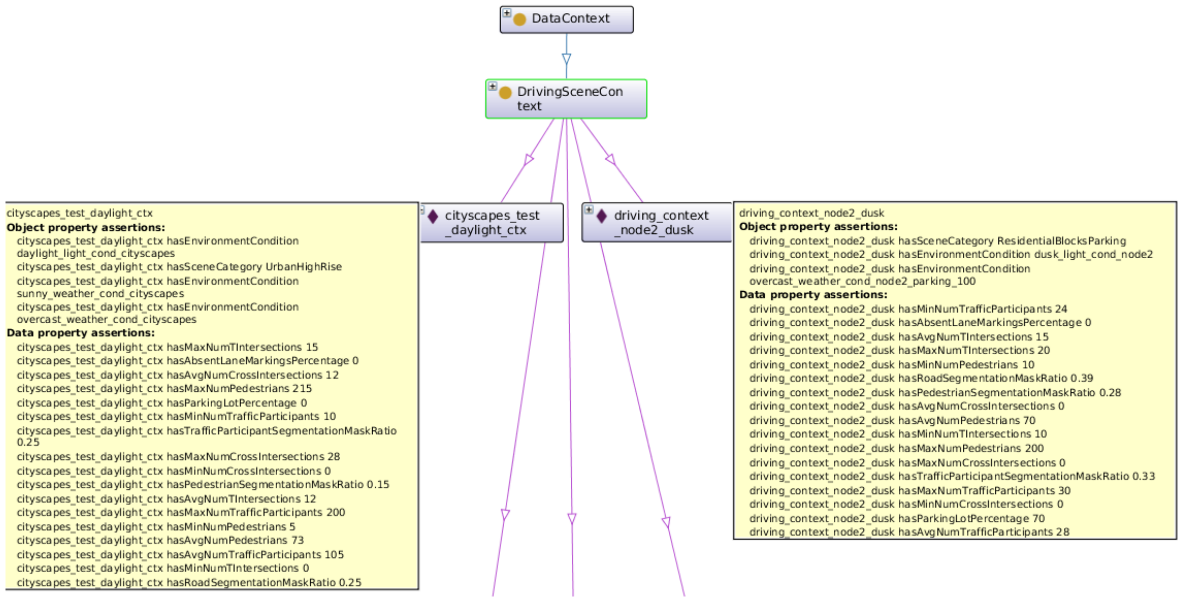
- The minimum, average and maximum number of pedestrians and other traffic participants
- The minimum, average and maximum number of cross or T-shaped intersections
- The average size ratio between a pedestrian or traffic participant and the whole image resolution (i.e. how large do pedestrians or traffic participants appear in an image)

The values to the properties are obtained using approaches as those described in Section 6.3. Depending on the values of these properties an agent can determine whether the current ML-based decision making model that it is currently using is still appropriate for the *data context* it is perceiving from its environment. If the model is no longer suitable from the *semantic* perspective described by the ontology, the agent will use the messaging protocol described in Section 3 (see also Figure 10).

6.3 Experiments and Results

Together with implementing the elements of the AI Folk project we have also developed experiments testing essential functionality in the project. The following functionality was tested:

- interaction between agents by means of the AI Folk protocol.
- functionality of the Ontology Driver for storing model descriptions and building the description of searched models.



```
cityscapes_test_daylight_ctx
Object property assertions:
  cityscapes_test_daylight_ctx hasEnvironmentCondition
  daylight_light_cond_cityscapes
  cityscapes_test_daylight_ctx hasSceneCategory UrbanHighRise
  cityscapes_test_daylight_ctx hasEnvironmentCondition
  sunny_weather_cond_cityscapes
  cityscapes_test_daylight_ctx hasEnvironmentCondition
  overcast_weather_cond_cityscapes
Data property assertions:
  cityscapes_test_daylight_ctx hasMaxNumTIntersections 15
  cityscapes_test_daylight_ctx hasAbsentLaneMarkingsPercentage 0
  cityscapes_test_daylight_ctx hasAvgNumCrossIntersections 12
  cityscapes_test_daylight_ctx hasMaxNumPedestrians 215
  cityscapes_test_daylight_ctx hasParkingLotPercentage 0
  cityscapes_test_daylight_ctx hasMinNumTrafficParticipants 10
  cityscapes_test_daylight_ctx hasTrafficParticipantSegmentationMaskRatio
  0.25
  cityscapes_test_daylight_ctx hasMaxNumCrossIntersections 28
  cityscapes_test_daylight_ctx hasMinNumCrossIntersections 0
  cityscapes_test_daylight_ctx hasPedestrianSegmentationMaskRatio 0.15
  cityscapes_test_daylight_ctx hasAvgNumTIntersections 12
  cityscapes_test_daylight_ctx hasMaxNumTrafficParticipants 200
  cityscapes_test_daylight_ctx hasMinNumPedestrians 5
  cityscapes_test_daylight_ctx hasAvgNumPedestrians 73
  cityscapes_test_daylight_ctx hasAvgNumTrafficParticipants 105
  cityscapes_test_daylight_ctx hasMinNumTIntersections 0
  cityscapes_test_daylight_ctx hasRoadSegmentationMaskRatio 0.25
```

```
driving_context_node2_dusk
Object property assertions:
  driving_context_node2_dusk hasSceneCategory ResidentialBlocksParking
  driving_context_node2_dusk hasEnvironmentCondition dusk_light_cond_node2
  driving_context_node2_dusk hasEnvironmentCondition
  overcast_weather_cond_node2_parking_100
Data property assertions:
  driving_context_node2_dusk hasMinNumTrafficParticipants 24
  driving_context_node2_dusk hasAbsentLaneMarkingsPercentage 0
  driving_context_node2_dusk hasAvgNumTIntersections 15
  driving_context_node2_dusk hasMaxNumTIntersections 20
  driving_context_node2_dusk hasMinNumPedestrians 10
  driving_context_node2_dusk hasRoadSegmentationMaskRatio 0.39
  driving_context_node2_dusk hasPedestrianSegmentationMaskRatio 0.28
  driving_context_node2_dusk hasAvgNumCrossIntersections 0
  driving_context_node2_dusk hasAvgNumPedestrians 70
  driving_context_node2_dusk hasMinNumTIntersections 10
  driving_context_node2_dusk hasMaxNumPedestrians 200
  driving_context_node2_dusk hasMaxNumCrossIntersections 0
  driving_context_node2_dusk hasTrafficParticipantSegmentationMaskRatio 0.33
  driving_context_node2_dusk hasMaxNumTrafficParticipants 30
  driving_context_node2_dusk hasMinNumCrossIntersections 0
  driving_context_node2_dusk hasParkingLotPercentage 70
  driving_context_node2_dusk hasAvgNumTrafficParticipants 28
```

Figure 9: View of instances of the DrivingSceneContext concept. The *cityscapes_test_daylight_ctx* individual identifies a data context from the Cityscapes dataset, while the *driving_context_node2_dusk* identifies an individual representing a data context from the scene encountered by an agent during runtime.

- ability of agents to use the ML Pipeline in order to obtain output from the received input
- ability of agents to decide which model is more appropriate for a given situation.
- ability of agents to recognize situation in terms of a given set of parameters (e.g. weather, number of pedestrians).
- ability to select the model to use in the ML Pipeline at runtime.

The concrete scenario that was researched was the changing of the currently used prediction model for segmentation, based on the appropriateness of the model in the context of a large number of pedestrians close to the autonomous vehicle. We have made this choice based on the comparison between the average percentage taken by the pedestrians in the images on which the model was trained.

Given our observations – presented in Figure 11, one can see that, in the case where more pedestrians appear in the scene, YOLO detects more pedestrians in both datasets, so it is more adequate for use in this case.

The tested scenario was the following: agent A is currently using the Deeplab model for segmentation. At a certain point, the number of pedestrians identified in a recent window of 5 images becomes considerably greater than the average of pedestrians in the dataset on which the model was trained. In this case, agent A looks for another model to use, if any is available. It will ask agents B and C for appropriate models. Agent B recommends the use of YOLO, which is more appropriate for the case of more pedestrians. Agent A downloads the model and switches to using it, at least for the period in which the number of pedestrians is high. A detailed view of the interaction between agents, drivers and sub-agent components is presented in Figure 10.

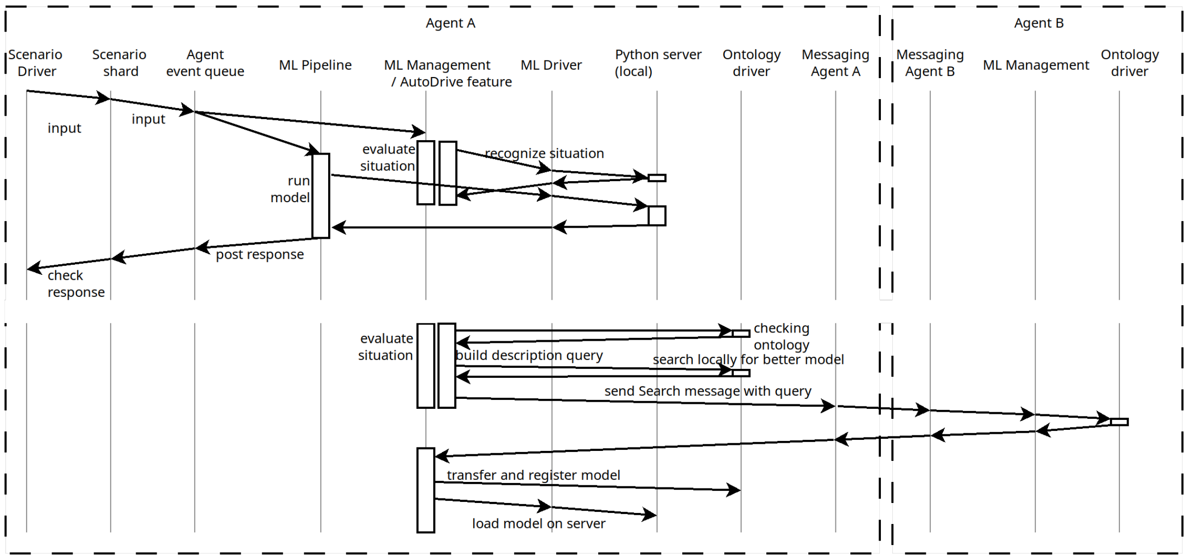


Figure 10: Interaction diagram between the various components in the implementation, when an input is received, and when the situation requires a more adequate model.

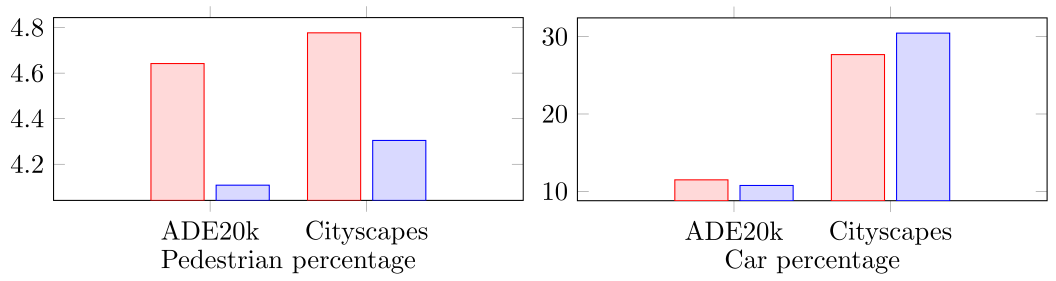


Figure 11: Comparison of average surface percentage of pedestrians and cars identified by YOLO (in red) and Deeplab (in blue) in the ADE20k and Cityscapes datasets.

6.4 Enhanced experiment design

To apply the methodology presented in Section 5 we design an experimentation protocol, describing the application conditions, the experiment steps and the evaluation method.

The AI Folk approach is highly relevant in situations where in the environment of an agent a *data distribution shift* is expected to occur sufficiently often. Our reference scenario of semantic segmentation in autonomous driving, implying a large mobility of the self-driving agent and the possibility to experience significant changes of driving scenes, is a good example of this application condition. To simulate such conditions from existing datasets, the following steps are followed.

First, a domain of application (e.g. autonomous driving) and a main task (e.g. drivable road segmentation) are specified. For the main task, attributes which can influence it are identified (e.g. min/average/max number of pedestrians encountered over a recent time window). This leads to an instance of the core AI Folk ontology expressing the data characteristics which are deemed to influence an ML algorithm by existing human domain knowledge.

Second, for the identified data attributes, a set of domain-knowledge informed *change conditions* are defined, whose objective it is to determine whether there is a *mismatch* between the characteristics of the data on which the currently employed ML model (which is solving the main task) was trained and the characteristics of the data from the currently experienced scenario (e.g. the minimum number of pedestrians detected over frames collected for the past minute is higher than the maximum number of pedestrians detected on any frame of the training set for the currently used segmentation model).

A detected mismatch signals the need to search for a replacement model, obtained from the *experience* of other agents in the community (see Section 5). Step three involves setting filtering criteria for searched models. New models need to have been trained on data that has *a suitable similarity* with respect to the characteristics of the attribute that caused the original mismatch. The filter is set up as a domain-knowledge informed reasoning rule that looks at the suitability of the data characteristics (e.g. the difference in average number of pedestrians is below a threshold), as well as the *performance* of the model on the main task (e.g. the pixel-wise accuracy for the segmentation of the drivable road class is above 95%).

The next step involves simulating the condition mismatch using one or more existing dataset(s) (e.g. Cityscapes or KITTI-360 for our segmentation task). A ML model (e.g. a DeepLab-v3 model) is finetuned on the main task on splits of data from a first selected dataset, which correspond to mismatching conditions on the identified attribute (e.g. number of pedestrians). A second ML model (e.g. YOLOv8) is finetuned on the same main task on a second dataset, on a split of data respecting the characteristics of the identified data attribute. Variation can be created in this case, by creating dataset splits that correspond to different characteristics of the selected data attribute (e.g. splits of data corresponding to different average pedestrian detection per 100 frames). Each finetuned model is given to an agent in the modeled *experience sharing* communities. Each agent has an evaluation holdout dataset.

To evaluate the experiment, agents run the mismatch identification rules and the corresponding search for a replacement model from the *experience* of other agents. The agent receives the most appropriate replacement model, based on the infrastructure and interactions described in Section 3. The current model and the replacement model are evaluated on the evaluation holdout dataset and if the percentage difference in the evaluation metric for the performance on the main task (e.g. pixel-wise accuracy / Dice-score for drivable road segmentation) shows, for all agents, a net increase in performance on the main task, then this constitutes an indication that reasoning on *domain knowledge informed mismatch conditions for key data characteristics* is a key proxy for anticipating performance improvement if a ML model were to be retrained on the current distribution shift. The main advantage of the AI Folk methodology is that, under the assumption of high data distribution shifts for high mobility of agents in their environment, reasoning over similar experiences in terms of domain-knowledge informed data characteristics is faster and more convenient than retraining or updating of the local ML model (as is typical in a federated learning setup).

7 Disaster Response Scenario

The disaster response scenario was constructed following the same structure as the autonomous driving scenario. The objective of having a second scenario was to show that, indeed, it is only necessary to change the scenario-specific parts of the deployment and the other components remain invariant. The validation was successful.

7.1 Datasets

FloodNet [BinaLab, 2021] is a dataset that provides high-resolution unmanned aircraft system imagery and detailed annotations regarding the damages. The data were collected using an unmanned aircraft system after Hurricane Harvey. The entire dataset consists of 2343 images that were divided into training (about 60% of the dataset), validation (around 20% of the dataset), and test (about 20%). The entire dataset is split into two tracks. The first track was for Semi-supervised Classification and Semantic Segmentation and contains 400 images and masks for training and 1050 unlabeled images. The second track was for Visual Question Answering and has 1450 images and a total of 4511 image-question pairs.

AIDER (Aerial Image Dataset for Emergency Response applications) [Kyrkou, 2020] is a dataset that was created manually by collecting images for four types of natural disasters Fire/Smoke, Flood, Collapsed Buildings and Traffic Accidents. This dataset also contains a class for the “Normal” case. The images from the dataset were gathered through more online sources such as Google images, Bing images, YouTube, and other websites. The dataset contains 500 images for each disaster mentioned earlier and also 4000 images for the “Normal” case .

CrisisMMD: Multimodal Crisis Dataset [CrisisNLP, nown] describes a dataset that contains several thousands of images collected from Twitter during seven major natural disasters such as floods, wildfires, earthquakes, and hurricanes. The dataset includes three labels: “Informative”, “Not informative”, “Don’t know or can’t judge”.

Flood Amateur Video for Semantic Segmentation Dataset [Naili, 2023] comprises images and videos from the social media network Instagram, captured during the flood in the city of Parepare, South Sulawesi Province. The dataset contains 6 labels for object classifications based on color: floods (blue light), buildings (red), plants (green), people (sage), vehicles (orange), and sky (dark blue). It is suitable for object recognition tasks using the semantic segmentation method.

xView [xViewOrg , 2018] presents one of the largest publicly available datasets of satellite imagery, containing images from complex scenes from different natural disaster situations, annotated using bounding boxes. The dataset includes multiple examples, with a total of 60 classes, such as damaged buildings, huts/tents, stadiums, fixed-wing aircraft, flooded areas, and more. xView also comes with a pretrained model that uses the TensorFlow object detection API .

Flood area segmentation [Faizal, 2023] describes a dataset containing 290 images of flooded areas and corresponding mask images that highlight the water regions. The masks for the images were created using Label Studio, which is an open-source data labeling software. The dataset includes a folder for the images, one for the masks, and a .CSV file that maps the image name with its mask.

Roadway flooding image dataset [Shahane, 2021] describes a dataset containing 441 annotated roadway flood images from coastal areas that experience frequent flooding due to storm surge, heavy rain, or sea level rise. The dataset includes two folders: one for the images and one for the masks.

7.2 Models

There are several already existing pretrained models and pipelines for training models such as:

xView2 [xViewOrg , 2018] with the main goal to provide quick and accurate information in case of a disaster. Before the search and rescue teams can go in the affected area they need to know the location and severity of the damage. This machine learning algorithm focuses on automating the process of assessing damage after a natural disaster. The xView organization started the project as a challenge for the community and they also released one of the largest datasets containing satellite imagery for natural disasters. This model was also used for disaster logistics and rescue missions on the ground in Turkey in 2023. For this algorithm UAVs imagery can be used in order to do a damage assessment on affected buildings because the drone imagery can provide higher resolution.

Semi-Supervised Classification and Segmentation on High Resolution Aerial Images [Khose, 2021] describes a pipeline containing two scripts: one for training an image classification model and one for

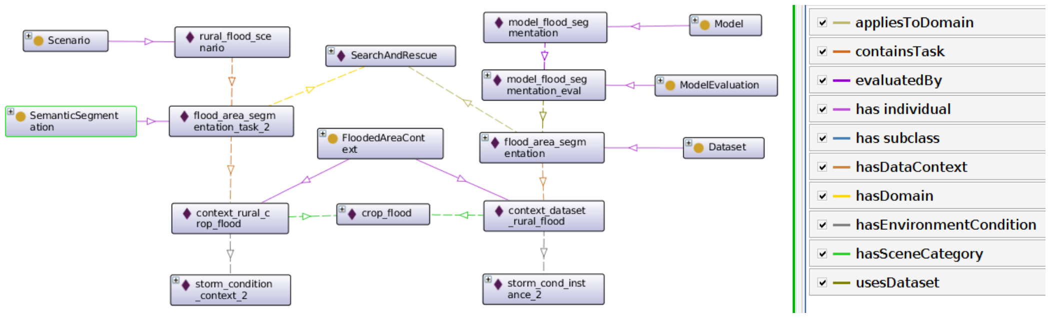
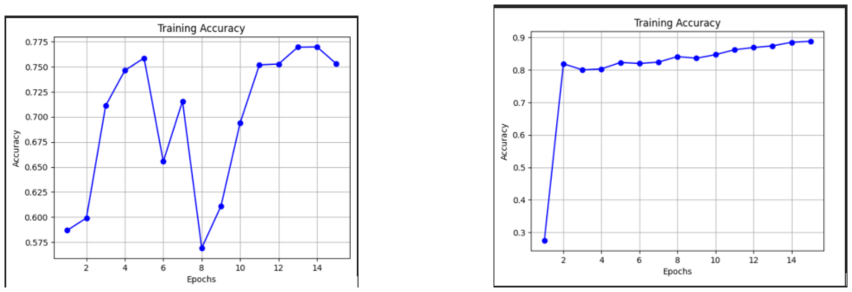


Figure 12: Concepts and relations in the ontology for the disaster response scenario



(a) (b)

Figure 13: Training accuracy for the Keras model (a) and for the UNet model (b).

training an image segmentation model made for the FloodNet challenge.

Aerial Image Semantic Segmentation [Ullah, 2023] describes a pipeline for fine-tuning a PyTorch pre-trained model InceptionV4 designed for image segmentation.

Aerial view segmentation [Ghinassi, 2022] describes a pipeline designed to train a U-Net model, with a ResNet50 backbone, for image segmentation.

Semantic Segmentation [Ruthvik, Sai Mondreti , 2024] describes a pipeline designed to train a U-Net model, with a ResNet50 backbone, for image segmentation.

Water Detection [Corredor, 2024] describes a pipeline for training a Keras model for flooded areas image segmentation.

7.3 Individual Results

	UNet Model	Keras Model
Flood Area Segmentation	0.0114	0.4291
Roadway Flooding Image	0.2937	0.5200

Table 3: Average IoU scores for UNet and Keras models

Figure 12 shows how the *DataContext* concept from the *core* AI Folk ontology is extended by a *DisasterSceneContext* specific one. For this scenario, we have emphasised the type of area where the flood takes place – in a crop or in a rural area.

For the project two models for image segmentation were trained using Semantic Segmentation and

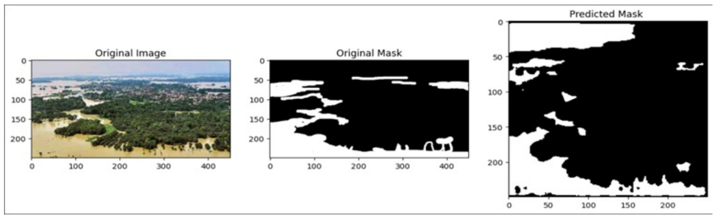


Figure 14: Mask Prediction of Keras model

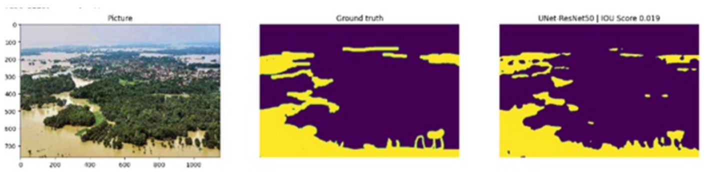


Figure 15: Mask prediction of UNet model

Water Detection pipelines, on the Flood Area Segmentation dataset.

Training results are presented in Figure 13. For both models, the training accuracy has increased, indicating that the models were trained properly. For validation, a split from the Flood Area Segmentation (43 images) and Roadway Flooding Image (67 images) datasets were used. To evaluate and compare the performance of each model for each image, an IoU score was calculated, and at the end, the average of IoU scores was calculated.

After comparing the results of the two models it is clear that the Keras models is more suitable for the task of detecting flooded areas then UNet with ResNet-50 backbone model. This happens because the Keras model is a custom model, tuned for detecting flooded areas, while the UNet with ResNet-50 backbone model is a general model used for image segmentation. The performance of the Keras model can also be higher because the images received as input are converted to grayscale and this can be more suitable for the task since converting to grayscale help to delimit the flooded area from the non-flooded one.

The validation scenario was similar to the autonomous driving scenario. Reaching an area with crop floods, a drone detects that the type of area has changed (there are a lower number of buildings in a crop than in a rural area) and finds a model more suitable for this type of segmentation.

8 Conclusion

In a world in which a large variety of machine learning prediction models exist for the same task, in intelligent software agent must be able to choose among the alternatives and must be able to interact with other agents and exchange information on the available models. The AI Folk projects has made several steps towards achieving this goal.

8.1 Project Achievements

The AI Folk project has made the following contributions:

- development of an approach for the management of ML prediction models in a community of software agents, independent of the concrete application domain;

- development and deployment of a software infrastructure, featuring a modular approach, clearly separating scenario-specific elements from scenario-independent components;
- specification of an interaction protocol to be used by a community of agents exchanging information about prediction models;
- development of a core ontology describing datasets and prediction models;
- development of the scenario-specific elements for two different application domains;
- development of a methodology for the application of the AI Folk approach to any application scenario.

8.2 Project Impact

The resources developed during the project are open-source and available online.

Three publications currently bear the project acknowledgement. a conference paper presented at the EMAS 2023 Workshop at AAMAS 2023 presented the underlying model of the infrastructure used for the AI Folk deployment, emphasizing on the formal foundation allowing the FLASH-MAS multi-agent framework to work with a variety of resources, including machine learning prediction model; a journal paper in Sensors, also in 2023, presents a distributed communication protocol – the Websocket Regions protocol – which can be used in decentralized, distributed deployments such as the ones envisaged by the AI Folk project and which serves as an underlying layer for the AI Folk interaction protocol; a conference paper presented at DCAI 2024 presents the AI Folk approach and methodology, based mainly on the autonomous driving scenario. It garnered relevant questions regarding the infrastructure and the methodology.

8.3 Future Work

There are two main directions for future development. One is further, more complex experimentation with AI Folk-based setups, as described in Section 6. This will give more complete results and will allow the testing of performance and the comparison between using smaller, more specialized models, and using larger, more capable, but more costly, models.

The second area of future development is the use of the AI Folk approach in novel types of scenarios, dealing with other types of machine learning resources. A submitted project proposal – "WAIz - Hypermedia-based agents using foundational models and knowledge exchange to create personalized plans" – posits that the AI Folk approach could be used to select among LLM prompt sets for planning in an energy-efficient smart building.

References

[Armeni et al., 2017] Armeni, I., Sax, A., Zamir, A. R., and Savarese, S. (2017). Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*.

[Bellifemine et al., 1999] Bellifemine, F., Poggi, A., and Rimassa, G. (1999). JADE - a FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, pages 97–108. Citeseer.

[BinaLab, 2021] BinaLab (2021). FloodNet Challenge EARTHVISION2021. https://github.com/BinaLab/FloodNet-Challenge-EARTHVISION2021.

[Brostow et al., 2009] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97.

[Cordts et al., 2016] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

[Corredor, 2024] Corredor, C. (2024). Water Detection. https://www.kaggle.com/code/carloscorredor24/water-detection/notebook.

[CrisisNLP, nown] CrisisNLP (Unknown). CrisisMMD: Multimodal Crisis Dataset. https://crisisnlp.qcri.org/crisismmd.html.

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338.

[Faizal, 2023] Faizal, K. (2023). Flood area segmentation. https://www.kaggle.com/datasets/faizalkarim/flood-area-segmentation.

[Geiger et al., 2013] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237.

[Ghinassi, 2022] Ghinassi, A. (2022). Aerial view segmentation. https://github.com/santurini/aerial-view-segmentation.

[Jocher et al., 2023] Jocher, G., Chaurasia, A., and Qiu, J. (2023). YOLO by Ultralytics.

[Khose, 2021] Khose, S. (2021). Semi-Supervised Classification and Segmentation on High Resolution Aerial Images. https://github.com/sahilkhose/FloodNet.

[Klampanos et al., 2019] Klampanos, I. A., Davvetas, A., Koukourikos, A., and Karkaletsis, V. (2019). Annett-o: an ontology for describing artificial neural network evaluation, topology and training. *International Journal of Metadata, Semantics and Ontologies*, 13(3):179–190.

[Kyrkou, 2020] Kyrkou, C. (2020). AIDER (Aerial Image Dataset for Emergency Response Applications). https://zenodo.org/records/3888300#.XvCPQUUzaUk.

[Li et al., 2020] Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60.

[Naili, 2023] Naili, S. I. (2023). Flood Amateur Video for Semantic Segmentation Dataset. https://data.mendeley.com/datasets/3kzr8mt8s2/4.

[Neuhold et al., 2017] Neuhold, G., Ollmann, T., Rota Bulo, S., and Kontschieder, P. (2017). The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 4990–4999.

[Nguyen and Weller, 2019] Nguyen, A. and Weller, T. (2019). Fairnets search-a prototype search service to find neural networks. In *SEMANTiCS Posters&Demos*.

[Olaru et al., 2023] Olaru, A., Nicolae, G., and Florea, A. M. (2023). The entity-operation model for practical multi-entity deployment. In Ciortea, A., Dastani, M., and Luo, J., editors, *Engineering Multi-Agent Systems*, pages 253–270, Cham. Springer Nature Switzerland.

[Olaru and Pricope, 2023] Olaru, A. and Pricope, M. (2023). Multi-modal decentralized interaction in multi-entity systems. *Sensors*, 23(6):3139.

[Olaru et al., 2019] Olaru, A., Sorici, A., and Florea, A. M. (2019). A flexible and lightweight agent deployment architecture. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 28-30 May 2019*, pages 251–258. IEEE.

[Ros et al., 2016] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. M. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243.

[Ruthvik, Sai Mondreti , 2024] Ruthvik, Sai Mondreti (2024). Semantic Segmentation. https://github.com/ruthviksai/SemanticSegmentation.

[Shahane, 2021] Shahane, S. (2021). Roadway flooding image dataset. https://www.kaggle.com/datasets/saurabhshahane/roadway-flooding-image-dataset/data.

[Shen et al., 2024] Shen, W., Li, C., Chen, H., Yan, M., Quan, X., Chen, H., Zhang, J., and Huang, F. (2024). Small llms are weak tool learners: A multi-LLM agent. *arXiv preprint arXiv:2401.07324*.

[Ullah, 2023] Ullah, M. (2023). Aerial Image Semantic Segmentation. https://github.com/bulentsiyah/semantic-drone-dataset.

[Wang et al., 2019] Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., and Chen, M. (2019). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5):156–165.

[Wojciechowski, 2001] Wojciechowski, P. T. (2001). Algorithms for location-independent communication between mobile agents. In *Proc. of AISB'01 Symposium on Software Mobility and Adaptive Behaviour*.

[Xiao et al., 2010] Xiao, J., Hays, J., Ehinger, K. A., Oliva, A., and Torralba, A. (2010). Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3485–3492. IEEE.

[xViewOrg , 2018] xViewOrg (2018). xView 2018 Detection Challenge. http://xviewdataset.org/.

[Yu et al., 2020] Yu, F., Chen, H., Wang, X., Xian, W., Chen, Y., Liu, F., Madhavan, V., and Darrell, T. (2020). Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645.

[Yurtkulu et al., 2019] Yurtkulu, S. C., Şahin, Y. H., and Unal, G. (2019). Semantic segmentation with extended deeplabv3 architecture. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE.

[Zhou et al., 2017] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., and Torralba, A. (2017). Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641.

A Competency questions for ML models

- General Dataset Information Questions
  1. What is the dataset name?
  2. What is the dataset access URL?
  3. What is the dataset human readable description?
  4. When was the dataset created?
  5. When was the dataset last updated?
- General Domain description Questions
  The *domain* is defined as an *intended principal / high-level* usage of the dataset (e.g. Autonomous Driving, Search and Rescue, Activity Recognition).
  A *Task* is defined as a *learnable objective* that can be addressed using the dataset (e.g. Semantic Segmentation of drivable road and driving obstacles, Semantic Segmentation of terrain types)
  1. What is the *domain* of the dataset?
  2. What *tasks* can be learned from the dataset?
- Semantic Segmentation Task Questions
  1. What *type* (e.g. image, 3D point-cloud) of input does the segmentation task have?
  2. What is the *size* (as tensor dimensions) of the segmentation task input?
  3. How many target classes does the segmentation task have? (e.g. drivable lane, opposite lane, traffic participant, pedestrian, other)
  4. What are the target classes for the segmentation task?
- Semantic Segmentation Task for Autonomous Driving
  Scene Categories define a highlevel classification of the driving scene (e.g. residential house-only, residential block-only, residential mixed, urban high-rise, single lane highway, multi lane highway, country side, mountain road)
  1. What are the illumination conditions for driving scenes present in the dataset? What is the percent-wise coverage of each illumination condition in the dataset?
  2. What are the weather conditions for driving scenes present in the dataset? What is the percent-wise coverage of each weather condition in the dataset?
  3. What are the *scene categories* present in the dataset? What is the percent-wise coverage of each *scene category* in the dataset?
  4. How many lanes per driving direction do roads in the dataset have?
  5. What is the min / max / avg number of traffic participant obstacles per frame?
  6. What is the min / max / avg number of pedestrians per frame?
  7. What is the min / max / avg number of cross intersections per frame?
  8. What is the min / max / avg number of T-shaped intersection per frame?
  9. What percentage of images in the dataset contain road section without any lane markings?
  10. What percentage of images in the dataset contain parking lot scenes?
  11. What is the min / max / avg *ratio* of the drivable road segmentation mask with respect to the input image resolution?
  12. What is the min / max / avg *ratio* of the perdestrian segmentation mask with respect to the input image resolution?
  13. What is the min / max / avg *ratio* of the traffic participant segmentation mask with respect to the input image resolution?

B Elements of the AI Folk Ontology

Table 4: Elements in the AI Folk ontology (continued in Table 5)

Problem description

1. **Category**
   - (a) Classification
   - (b) Regression
   - (c) Segmentation
   - (d) ObjectDetection
   - (e) Clustering
   - (f) Adversarial
   - (g) Discrimination
   - (h) Generation
   - (i) Prediction
   - (j) Estimation
   - (k) Reconstruction
2. **EvaluationMetric**
3. **DataModality**

Dataset description

1. Name
2. URL
3. CreationTime
4. Domain
5. Application/ possible tasks
6. Number of images
7. Input type (video, image, lidar etc)
8. Number of classes (if applicable)
9. Light conditions
10. Weather conditions
11. Area recorded
12. Number of traffic participants
13. Number of cars
14. Number of pedestrians
15. Keywords
16. Citation
17. Size

Model description (A)

ANNArchitecture:

1. PretrainedNetwork
2. InputSize
3. OutputSize
4. Layer
   - (a) ConvolutionLayer
     - i. Conv1d
     - ii. Conv2d
     - iii. Conv3d
     - iv. ConvTranspose1d
     - v. ConvTranspose2d
     - vi. ConvTranspose3d
     - vii. Unfold
     - viii. Fold
   - (b) Poolinglayer
     - i. MaxPool1d
     - ii. MaxPool2d
     - iii. MaxPool3d
     - iv. MaxUnpool1d
     - v. MaxUnpool2d
     - vi. MaxUnpool3d
     - vii. AvgPool1d
     - viii. AvgPool2d
     - ix. AvgPool3d
     - x. AdaptiveMaxPool1d
     - xi. AdaptiveMaxPool2d
     - xii. AdaptiveMaxPool3d
   - (c) PaddingLayer
     - i. ReflectionPad1d
     - ii. ReflectionPad2d
     - iii. ReflectionPad3d
     - iv. ReplicationPad1d
     - v. ReplicationPad2d
     - vi. ReplicationPad3d
     - vii. ZeroPad2d
     - viii. ConstantPad1d
     - ix. ConstantPad2d
     - x. ConstantPad3d

Table 5: Elements in the AI Folk ontology (continued from Table 4)

Requirements	Model description (B)
1. Operating System 2. Language 3. Has Installation Script (to be executed instantly) 4. Dependent libraries 5. MLFramework 6. ModelSize 7. GPU 8. RAM size 9. Memory size	(d). NormalizationLayer (i) BatchNorm1d (ii) BatchNorm2d (iii) BatchNorm3d (iv) GroupNorm (v) InstanceNorm1d (vi) InstanceNorm2d (vii) InstanceNorm3d (e). RecurrentLayer (i) RNN (ii) LSTM (iii) GRU (iv) RNNCell (v) LSTMCell (vi) GRUCell (f). TransformerLayer (i) Transformer (ii) TransformerEncoder (iii) TransformerDecoder (iv) TransformerEncoderLayer (g). LinearLayer (i) Identity (ii) Linear (iii) Bilinear (h). DropoutLayer (i) Dropout (ii) Dropout1d (iii) Dropout2d (iv) Dropout3d (i). SparseLayer (i) Embedding (ii) EmbeddingBag (j). VisionLayer (i) PixelShuffle (ii) PixelUnshuffle (iii) Upsample (iv) UpsamplingNearest2d (v) UpsamplingBilinear2d

C Python server available operations

- `/add_model` – The `add_model` service is a Flask route defined in the ML server. This service is designed to handle HTTP POST requests for adding machine learning models to the server. Here's a breakdown of how it works:
  1. Request Parameters
     The service expects the following parameters to be included in the POST request form data:
     - `model_name`: The name of the model.
     - `model_file`: The path to the model file.
     - `model_config`: A JSON-formatted string containing additional configuration for the model.
  2. Functionality
     Upon receiving a request, the service extracts the relevant information from the form data. It then attempts to load the model using the `get_model` function, which utilizes PyTorch to load the model from the specified file path. The loaded model is associated with its configuration, including the model name, file path, CUDA availability, task type, and dataset information. The loaded model is added to a global dictionary `models`, where model names serve as keys.
  3. Response
     The service returns a JSON response:
     - If the model is successfully added, it returns a success message indicating that the model has been added.
     - If there is an error (e.g., missing model name or model file), it returns an error message along with a 400 Bad Request status.
  4. Example POST Request Data

     ```
     {
       "model_name": "example_model",
       "model_file": "/path/to/example_model.pth",
       "model_config": "{\"cuda\": true, \"task\": \"classification\",
                        \"dataset\": \"mnist\"}"
     }
     ```

  5. Example Success Response

     ```
     {
       "message": "Model 'example_model' has been successfully added."
     }
     ```

     This service facilitates the dynamic addition of machine learning models to the server, allowing users to extend the set of available models without modifying the code.
- `/add_dataset`
  1. Request Parameters
     The service expects the following parameters to be included in the POST request form data:
     - `dataset_name`: The name of the dataset.
     - `dataset_classes`: A JSON-formatted string containing the class names for the dataset.
  2. Functionality
     Upon receiving a request, the service extracts the relevant information from the form data. It then checks if the dataset already exists in the global `datasets` dictionary. If it does not exist, the service adds the dataset with its class names to `datasets`.
  3. Response
     The service returns a JSON response:
     - If the dataset is successfully added, it returns a success message indicating that the dataset has been added.
     - If there is an error (e.g., the dataset already exists or missing dataset name or class names), it returns an error message along with a 404 Not Found status.

4. Example POST Request Data

   ```
   {
     "dataset_name": "example_dataset",
     "dataset_classes": "[\"class1\", \"class2\", \"class3\"]"
   }
   ```

5. Example Success Response

   ```
   {
     "message": "Dataset 'example_dataset' has been successfully added."
   }
   ```

   This service facilitates the dynamic addition of datasets to the server, allowing users to extend the set of available datasets without modifying the code.

- `/predict`
  1. Request Parameters
     The service expects the following parameters to be included in the POST request form data:
     - `model_name`: The name of the machine learning model.
     - `input_data`: Base64-encoded input data for prediction.
  2. Functionality
     Upon receiving a request, the service extracts the relevant information from the form data. It checks if the specified model exists in the global `models` dictionary. If the model exists, it performs the following steps:
     - Decodes the Base64-encoded input data.
     - Utilizes the PyTorch model and associated transformations to make a prediction.
     - Constructs a response containing the prediction results and additional information like class names, task type, and dataset.
  3. Response
     The service returns a JSON response:
     - If the model exists and the prediction is successful, it returns a response containing the prediction results and related information.
     - If there is an error (e.g., the specified model does not exist), it returns an error message along with a 404 Not Found status.
  4. Example POST Request Data

     ```
     {
       "model_name": "example_model",
       "input_data": "base64_encoded_data"
     }
     ```

  5. Example Success Response

     ```
     {
       "prediction": [0.85, 0.15, 0.0],
       "class_names": ["class1", "class2", "class3"],
       "task": "classification",
       "dataset": "example_dataset"
     }
     ```

     This service facilitates making predictions using a specified machine learning model, providing flexibility for different models and tasks.

- `/get_models`
  1. Functionality
     Upon receiving a request, the service reads the model configuration file specified by the constant `MODEL_CONFIG_FILE`. It then constructs a response containing information about available models by iterating through the models defined in the configuration.
  2. Response
     The service returns a JSON response:
     - If the model configuration file is successfully loaded, it returns a response containing a dictionary of available models and their respective configurations.

3. Example Success Response

   ```
   {
     "models": {
       "example_model1": {
         "name": "example_model1",
         "path": "/path/to/example_model1.pth",
         "cuda": true,
         "task": "classification",
         "dataset": "mnist"
       },
       "example_model2": {
         "name": "example_model2",
         "path": "/path/to/example_model2.pth",
         "cuda": false,
         "task": "regression",
         "dataset": "cifar10"
       },
       ...
     }
   }
   ```

   This service provides information about the available machine learning models based on the configuration file.

- `/export_model`
  1. Request Parameters
     The service expects the following parameters to be included in the POST request form data:
     - `model_name`: The name of the machine learning model to export.
     - `export_directory_path`: The path to the directory where the model will be exported.
  2. Functionality
     Upon receiving a request, the service checks if the specified model exists in the global `models` dictionary. If the model exists, it performs the following steps:
     - Reads the model configuration file specified by the constant `MODEL_CONFIG_FILE`.
     - Identifies the configuration for the specified model.
     - Creates a new configuration file containing only the information for the specified model.
     - Saves the new configuration file in the export directory.
     - Copies the model file to the export directory.
  3. Response The service returns a JSON response:
     - If the model exists and the export is successful, it returns a response containing a success message and the destination directory.
     - If there is an error (e.g., the specified model does not exist), it returns an error message along with a 404 Not Found status.
  4. Example POST Request Data

     ```
     {
       "model_name": "example_model",
       "export_directory_path": "/path/to/export_directory"
     }
     ```

  5. Example Success Response

     ```
     {
       "message": "Model 'example_model' has been successfully exported.",
       "destination": "/path/to/export_directory"
     }
     ```

     This service facilitates exporting a machine learning model along with its configuration to a specified directory.