Policy Enabled Caching for Distributed AI

Dinesh C. Verma
IBM T. J. Watson Research Center
Yorktown Heights, NY, U.S.A.
dverma@us.ibm.com

Graham Bent
IBM Research UK
Warrington, UK
GBent@uk.ibm.com

***Abstract*—Web Caching has established itself as a key enabling technology within the Internet. It enables efficient browsing of websites and web-based services on networks that are bandwidth constrained. However, similar techniques are not available for AI based solutions. Many AI solutions are based on deep neural networks or similar approaches which require creation of machine learning models trained with huge amounts of data. Such models are best created in centralized locations with significant processing power. In many environments, sending the data to a centralized location is infeasible or undesirable. A judicious combination of ideas borrowed from web-caching paradigm, with ideas from AI and machine learning can provide an effective solution for exploitation of deep learning models in bandwidth constrained environments. Allowing such caches to generate their own policies using a generative policy approach can enable the creation of a generic edge caching system which can be used with a wide variety of backend AI systems.**

Keywords—Edge Computing, Fog Computing, Distributed AI, Semantic Caching, Generative Policy

I. INTRODUCTION

The immense strides made in the field of Artificial Intelligence have spurred a significant interest in the industry in the various applications of AI and it is widely recognized that AI can transform industries. However, the transformative effect of AI also comes with the ability to have access to big data for training purposes. Having a large amount of training data enables us to create deep neural networks [1], and such networks can be very accurate in their predictions for domains such as speech processing [2] provided they have been trained with adequate data. Such training requires significant processing power [3] including use of hardware accelerators such as Graphics Processors, which can only be obtained in large-scale cloud deployments or large private data centers.

When these models are used for the inference step after training, the data needs to be sent to the same location where the model is, namely the central location where the model was created. However, in environments which have poor network connectivity, the access to such a site may make an AI solution impractical. It would be more effective to perform its inference step without sending the data to the central location. However, moving the model out of the training location to an external site may not always be feasible. Some of the models could be too large to move out of the training locations, e.g. deep neural networks models exceeding 6GBytes in size have been presented in the literature [4]. The network transfer time of such models would be prohibitively long when connecting over cellular or satellite networks. In other cases, the models may have dependencies on local resources or services, which can make exporting the model to another location difficult.

In this paper, we discuss how a policy driven approach can be used to determine the best mode in which edge sites can deal with the complexities involved in the model export process. The assumption would be that the model will be trained at a central location, and then exported to an edge-site. Policies would allow the approach to work even under conditions where extraction of models may not be simple.

In section II of this paper, we present some scenarios where AI approaches can be used and argue for using an edge based approach for them. In Section III, we present approaches for operation of edge sites, followed by a discussion on how the edge caches can generate policies to drive their operation. Finally, we provide the conclusions of our work and discuss future research directions.

II. SCENARIOS MOTIVATING DISTRIBUTED AI

Many AI applications in context of IoT can be viewed as a system that works off a model that embodies knowledge. The abstract representation of such an AI application is shown in Figure 1. The knowledge could be either encoded by a human being into a machine interpretable model, or a model could be built in an automated manner on the basis of some training data.

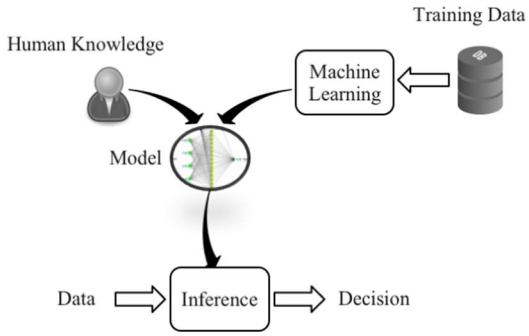

Fig. 1. Abstract Representation of an AI application

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Rule based systems, decision trees, and expert systems are some of the common models used for human knowledge representations. Neural networks and their various variants are a wide-spread approach to create models using machine learning on training data. Once the model is created, it can be used for the inference step, in which the model is used to map input data into a decision. As an example, a speech to text translator can create models by using a training data with several instances of speech and its corresponding text, create a model, and then use the model in the inference step to translate an input speech signal (data) to an output text (decision).

The process of training usually requires a significant amount of training data, and is usually best done in a central location where substantial processing power is available. Even when embodying human knowledge, it is better to create the human knowledge in a single location so that it can be stored, updated and distributed easily. The inference step is not similarly constrained, and it can be done in a different location. In many situations, it may be better for the inference step to be done in a location different than the one in which the model is built, learnt or defined. Some of those scenarios are described below.

A. Drone Intelligence

Drones have a tremendous potential for use in a military environment – from intelligence gathering, to information and kinetic warfare. They have an equally diverse and interesting set of applications in the civilian domain. However, their current operational model remains that of a relatively dumb data collection system that is controlled from a backend system, and uploads its data to the backend site for analytics and processing. This introduces a significant point of vulnerability since the drone's connectivity to the back-end control system can be attacked or jammed, rendering a swarm of drones dysfunctional and vulnerable. In an ideal world, the drones would have artificial intelligence capabilities, could learn themselves from the information that is acquired from their sensors, and take intelligent actions based on that information autonomously.

If a drone were doing a task like surveillance and needed to have a capability to recognize items like a lost person or hostile vehicles within an image, the training task requires a system

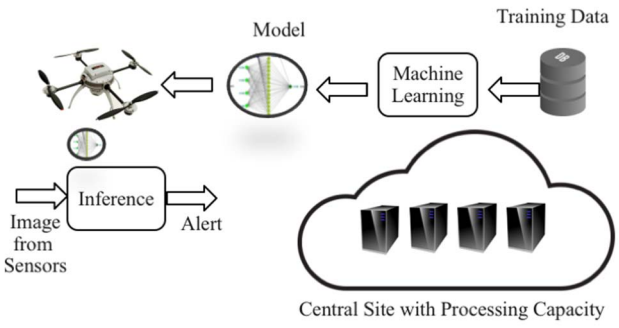

Fig. 2. Drone Intelligence

with processing capacity far beyond what is available on the drone. Such training is best done in a central cloud or data center infrastructure. However, when the drone is actually conducting a surveillance mission, it will be difficult to send collected images to the backend site in real-time where the model was

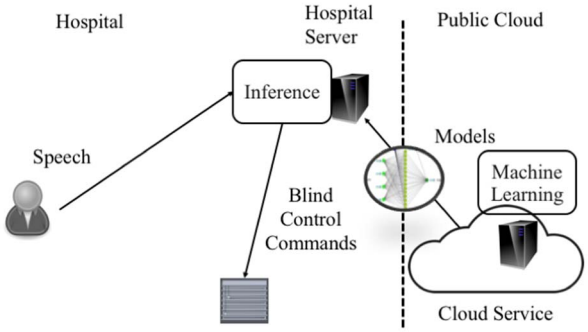

Fig. 3. Speech to Text in HealthCare

built because of bandwidth limitations. The inference operation would be best done on the drone itself, and an alert can go over to the relevant party when an interesting object is seen. The action to be undertaken when the alert is generated depends on the nature of the surveillance mission. As an example, if the drone is searching for a lost hiker, the alert would send the information to a rescuer at a central site, who can manually check if the drone has found the lost person, and ask the drone to drop a leaflet with instructions to await rescue as well as some food/water for the hiker. The setup for edge operation is shown in Figure 2.

B. Speech to Text in Commerical Environments

Speech Recognition is a very useful technology and manifests itself in devices such as Amazon Echo and Google Home, or in applications such as Apple Siri. In all of the speech recognition systems that are available today, the approach is for the speech signals to be sent to a cloud based service operated by the provider of the device/application (Amazon, Google or Apple). In the cloud based service, the provider has pre-trained models that convert the speech signal to text, and can perform additional actions based on that command.

In many commercial situations, however, it would be unacceptable for the speech signals to go to the cloud due to regulatory or security concerns. As an example, a hospital is bound to maintain privacy about the health information of a patient and conversations between a patient and a doctor should not be sent to the cloud system. Regulatory compliance has been known to slow the adoption of cloud in healthcare industry [5].

While it will be convenient for hospital staff and patients to use voice commands to control things in the hospital, e.g., lower or raise window blinds, the convenience feature can run afoul of healthcare regulations. Devices that continuously monitor the sounds in a room and send it to a cloud based service will not be acceptable for use within a hospital environment, since conversations between doctors and patients are confidential, subject to regulatory compliance, and can only be sent to cloud services that meet strict compliance standards.

Creating a private version of the system that works in a separate installation on hospital premises creates operational challenges. Models trained for speech to text are not the result of one-time training, and need to be continuously updated to add new accents, new words that are added to the language, new languages etc. This management and training requires

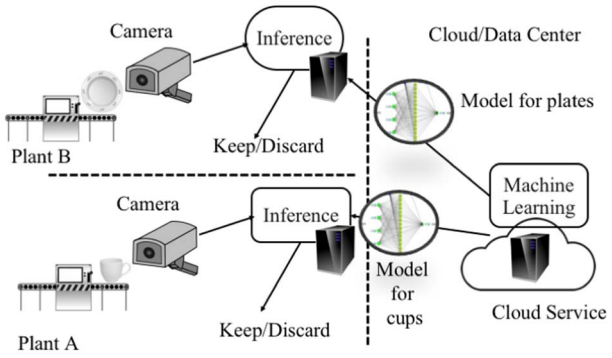

Fig. 4. Visual Inspection in Manufacturing

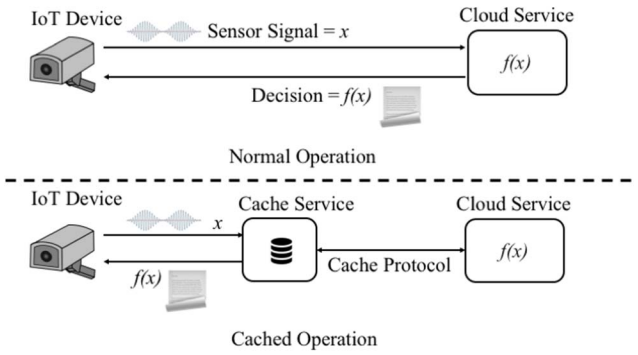

Fig. 5. Abstracted Representation of Caching System

significant complexity, computational horse-power and AI model-building expertise. Therefore, the hospital has to determine the trade-off between managing its own environment versus benefitting from the continuous model updates that can happen from a cloud managed resource.

A viable approach would be to have the speech to text system run the inferencing part on a machine within the hospital premises, while the cloud service is used to train and maintain the model. With this approach, the system maintains all conversations within the premises itself, while it can continuously get updated models for speech to text conversion. This model is shown in Figure 3. This gives it the best of both worlds, on-premises speech to text translation and conformance to policies, while it gets the advantage of continuous updates and training to the model as they are maintained by the cloud service.

While the analysis above has been presented in the context of a hospital, there are analogues of the same situation in many different commercial contexts. Several modern whiteboards and smart-screens are coming equipped with speech to text capability so that conversations can be transcribed as they are happening. Many businesses will be sensitive about sending their business conversations to external cloud based services, yet the benefits of cloud based service, having the ability to upgrade to different accents and different languages, are hard to replicate on a private on-premises infrastructure. In almost any commercial application of speech to text, concerns about maintaining confidentiality would lead to similar situations where the idea solution would be to run the inference step on-premises, but leave the task of training for new accents and languages to a central cloud location.

C. *Visual Inspection in Manufacturing*

In any manufacturing plant, products coming off the assembly line need to be inspected for defects. In many products, the defects can be identified visually by a camera. A image recognition model can identify good products from bad products, provided it is trained previously with several pictures showing the good products, as well as the most common defects that one may encounter in the products.

The same manufacturer may operate several manufacturing plants, and the products that are produced in the different plants may not be similar. The same manufacturer may operate one plant that produces ceramic plates, and another plan that produces ceramic cups. Using conventional image analysis techniques, different models will need to be created for ceramic cups and ceramic plates. Since image training requires a significant amount of resources, it is best done in a central location with adequate horse-power.

At the same time, the inference is best done at each of the different plants, both because this avoids sending lots of images to the network, but also because each plant can customize its inference process to best match the products it produces. The setup would be as shown in Figure 4. It shows two plants, one making cups and one making plates. The cloud service is used to train the models, while a small computer at each site can determine whether a newly produced item is good or defective. The same scenario should manifest itself in many different contexts, especially in IoT environments [6].

III. MODES FOR EDGE OPERATION

The three examples given in Section II are illustrative of the fact that inference can be done better in many situations in a location which is different than the large system used to train the AI or machine learning model. Without loss of generality, we refer to the sites where inference is being done as edge sites, and the locations where the training is done as cloud sites. The cloud site in itself is fully capable of doing both the inference and the training/learning part, but the edge site can only do the inference operation.

In general, there will be many edge sites as well as many cloud sites in any given scenario. The edge sites may need to determine what type of models they need to get from the cloud site, and also how they need to perform the required operations. The cloud service or the centralized service discussed in the previous section can be abstracted as a service which takes in an input which represents the sensor signal and produces and output which produces a decision.

While the field of general AI is very broad, and covers several areas such as reasoning, planning, learning etc. the discussion of the scenarios in the previous section illustrates that a large number of AI use-cases in IoT context is that of decision making. An input is obtained from one or more sensors, an AI model is consulted, and a decision is returned. While the AI model may be based on several sophisticated techniques, we can view it as an implementation of a function that translates the sensor signal into a decision. Our goal is to create an equivalent capability at an edge site, which would be able to perform the same translation from sensor signal to decision at the edge itself.

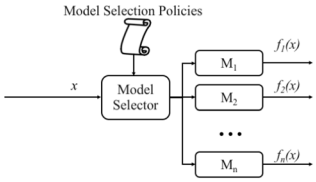

Fig. 6. Inference Logic at Centralized Service

The abstracted view of the caching system is shown in Figure 5. The cloud service is modeled as a function which takes the sensor signal *x* and returns the value of *f(x)*. The cache service sits in the network path between the IoT device and the cloud service, intercepts the message and returns the same function. The cache protocol is used by the cache service to ensure that it is working in coordination with the cloud service, and that the correct answer is being returned.

For most practical IoT scenarios, there will likely be more than one model being used at the cloud service. For drone intelligence use-case, the drone needs to identify objects in the environment. It will likely use one model to identify people, one model to identify wild animals, one model to identify a vehicle, another model to identify spots on fire etc. When the drone mission is underway, it may use all or some of those models to determine whether it needs to send an alert to a human observer. For speech to text, different models are needed for different accents of speakers and for different languages. For visual inspection in manufacturing and defect classification, there may be a separate model for each product or for each type of defect per product. While one can generalize that a simple composite model should be used for each application, it is quite likely that several models will be used, and the list of models will keep on increasing as new training data become available.

There are three general situations that may arise when operating in the mode shown in Figure 5. The cache may be able to replicate all the AI models that are being used in the cloud service, the cache may make itself more efficient by replicating only some of the AI models that are used in the cloud, or the cache may be unable to get the models out of the cloud. Depending on the option that is possible, the edge sites needs be configured to operate in one of three modes. These modes are described below.

A. Model Retrieval Mode

In one mode of operation, the edge may retrieve all of the models that the cloud site, or cloud sites may have trained, and use them to perform the inference operation. When the data input is received, it can use a set of policies to determine the type of model that needs to be used for the specific input. In a scenario like the surveillance by drones, multiple inputs may be received from the different sensors that are in the drone. The inference operation at the drone would need to determine which of the different models ought to be used with the specific sensor input.

One possible implementation of the model retrieval mode is shown in Figure 6. At the cloud service, several models are available due to previous training. The first step at the cloud service is to select the model that should be used for the request. The selection may be based on the identity of the requesting

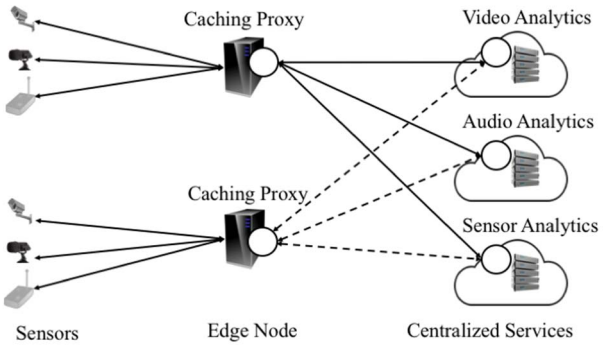

Fig. 7. Physical Toplogy of Caching Proxies and Cloud Services

client, fields specified in the request asking for a specific type of model to be used, or a combination thereof. Subsequently, the right model is chosen, and used to determine the decision for the provided input.

In order to retrieve the model, the caching protocol between the cache at the edge and the cloud service needs to provide a mechanism for the edge to retrieve and store those models. In order to enable this, the cloud service would need to export an interface which allows the cache to ask for inquire about available models, and then retrieve all of those models. It can also retrieve the policies that are used to select the right module to use on the incoming request.

If we assume that cloud service is a REST based service which accepts JSON request containing the sensor input and returns a JSON result containing the decision, then one can create a single proxy service that accepts any JSON request from the client, and using the cached policies and model selector, it can perform the transformation from the input request to the outbound requests at the proxy site itself. A single common caching logic can be used at the proxy site, and the same proxy site should be able service many different cloud services. As a result, the physical topology of caching proxies and centralized AI services will look like the one shown in Figure 7.

Model retrieval may be a good approach to use in environments where the models are small, and all of the models can be stored at the edge site. It is also a good approach to use when the operation mode requires that the edge site be disconnected from the cloud site for extended periods of time. Specifically, for the speech to text scenario described previously, where the models tend to be small for any accent-language combinations, retrieving all speech models at the edge from the cloud may be quite feasible. If a hospital wants to support the languages of Spanish and English, with the most common five accents for both languages, it would be able to retrieve all 10 models required for speech processing and run them in the cache.

B. Model Selection Mode

In some cases, the edge may have to deal with a situation where there may be too many models to use. In multi-cultural, multi-language, multi-ethnic environments found in any metropolitan area, there will be a need for hospitals to support ~100 languages and ~10 accents for each language. Maintaining and updating the models for all of these language accent

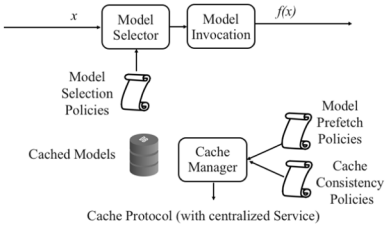

Fig. 8. Structure of Caching system at Edge Site

combinations can become a daunting task for most premises, and it is best to use them from a central cloud based service. At the same time, not all language and accents are needed at any time in any hospital, and the system in any hospital only needs to deal with a small subset of accents/languages at any time, depending on the patients they have on board.

As another example, when visual recognition on images needs to be done, detection of each type of object may use a different model. The model that defects in images of a cup is different than that which detects defects in plates, and one may even need a different model depending on the patterns used on cups/plates.

The large number of models that are required for different aspects of visual recognition may not all be useful to an edge site for their operation. The edge site would need to determine which models are best suited for its operation and only need to selectively retrieve the right set of models for its operation. In order to determine the set of models that it needs, it may rely on some policies that determine which models are to be retrieved, and which models ought to be removed because they do not appear to be relevant.

While in the model retrieval mode, the caching proxy replicated and cached all the model selection policies and the models at the local site, the caching proxy needs to add another set of policies, determining which models to cache and which ones not to cache into its system. While the cache can be made self-configuring by adopting a most recently used policy, and retrieve only the models that are going to be used, that approach has a disadvantage when a cache miss occurs. Retrieving the model and configuring the local system to use the model can be a time-consuming operation. Therefore, a decision needs to be made as to which set of models are the right ones to retrieve and use locally.

A policy driven approach can help the edge caching system to become more efficient when all models need not be kept within the edge site itself. The edge caching site in this case can be designed as operating according to the structure shown in Figure 8. A set of model pre-fetch policies and cache consistency policies are used to determine how the cache protocol ought to operate between the edge site and the cloud. These policies determine which subset of the AI models ought to be brought over to the edge site for its operation.

C. Semantic Caching Mode

In some cases, the model maintained at the cloud site may be too large to permit retrieval at the edge site. In other cases, the decision returned by the cloud may depend on resources or services that are present only in the cloud, and it may be

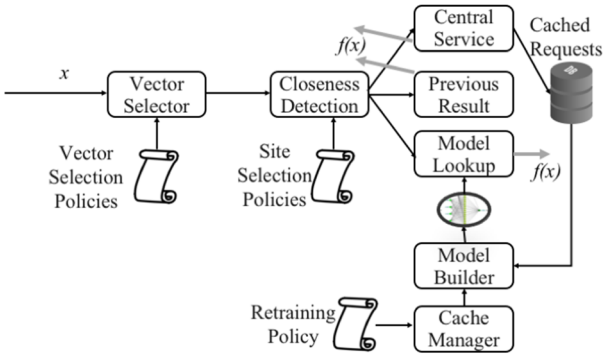

Fig. 9. Logical Architecture for Semantic Caching

impractical to move the decision making in those cases to the edge caching site. Model retrieval or model caching will not work in those cases, since the model can not to be mode to the edge. In those cases, the edge site would need to operate in a different mode, which we call the semantic caching mode.

The semantic caching mode can be viewed as an analog of the web-caching proxies popularly used with web-servers. Web caches work by storing the results of the requests made for previous URLs (Universal Resource Locators) by their clients. If a client makes the same request within a short period of time, the proxy can return the previously seen response. Usually, caching proxies check if the content referenced by a URL has changed from the server, an operation that can be performed using a substantially smaller amount of network data than retrieving the entire content once again.

The idea of semantic caching is to implement a similar function, but AI routines usually do not have a very cleanly defined URL like web-requests. In most common requests, an image or sound signal will be received, and it is rare for two images to be exactly identical. However, it is possible to determine if two signals are reasonably similar. One approach to check for similarity is to map the input into a vector (e.g. variations of word2vec [7]) and use the cosine-similarity as a measure of how close two requests are. If the requests are similar enough then the previously seen response can be returned. Decisions regarding the choice of the appropriate vector matching algorithm to use can be made using policies. The selection depends on the type of signal, the identity of the source of the signal, and the context in which the signal is being used.

In addition to determining closeness to the requests, the cache-misses, where the response comes from the centralized service, provide an alternative approach to create the semantic cache. Once a sufficient number of responses have been obtained from the cloud, a shallow model can be trained at the edge. The shallow model uses the responses from the backend cloud service to train a local AI model. If the shallow model returns an answer with a low confidence, the cloud site can be consulted again to provide an authoritative response. To use this approach, policies are needed which would allow the edge to determine when to retrain the model, and when the system ought to be accepting the response of the shallow model versus going back to the centralized service.

The policy driven structure of the logic at the cache is shown in Figure 9. When the request is first seen, a set of vector selection policies are used to determine which similarity measure to use for the incoming request. The similarity vector is used to determine if the request is close enough to an existing request in the cache or not. Another set of policies, the site selection policies, is used to determine if (a) the request should just return one of the previous responses since it is close enough (b) the request should use the local trained model or (c) the request should be forwarded to the central service. One of these is used to provide the response $f(x)$ to the input request. The building of the local model is also driven by means of retraining polices which determine when the local model should be built.

The semantic cache approach works best when the cache has connectivity to the cloud. The connectivity may be limited or expensive, but the cache can use it for occasionally reaching back to the cloud.

IV. Generative Policy Architecture

In order to drive the operation of the cache, and indeed to select the mode of cache operation, several policies are needed. In the normal mode of operation, the policies would be specified by a human operator. However, it would be much easier if the caches could determine their policies on their own. One of the ways in which caches can determine their own policies is to use a generative policy architecture, as shown in Figure 10.

The generative policy architecture [8] provides an approach by which a device can automatically generate its own policies. In this architecture, the human manager (via a management system) provides two types of information to each device. One is the representation of an interaction graph, and the other is a policy generator. A policy generator could be a template for policies, or it could be a grammar generating a language that represents the desired policies.

An interaction graph is an abstract description of the various entities within the environment that the device needs to interact with. The interaction graph is defined as a relationship between entities in different roles in the system, not as an exhaustive listing of all the different devices in the system. The role of each device in the interaction graph is defined by the manager.

Each device receives the interaction graph from the management system, and uses it to discover the other devices in the system with which it should connect. The interaction graph also provides the attributes of the connecting devices that affect the interactions between the managed devices, and any attributes of the links between a pair of managed devices. The device uses this information along with the generator to generate its local policies.

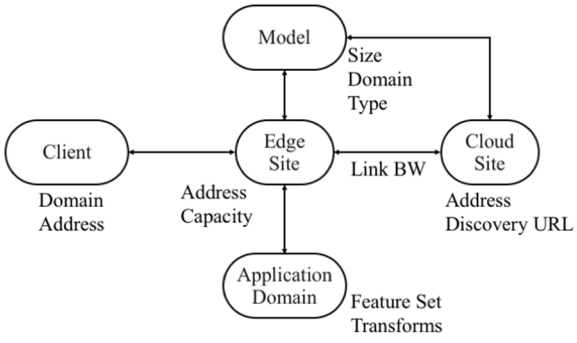

Fig. 11. Interaction Graph for Edge Sites to generate their own policies

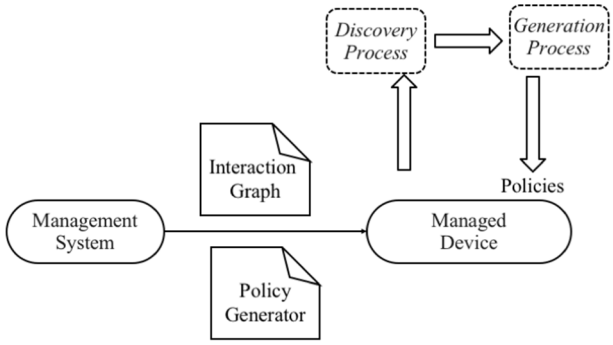

Fig. 10. Generative Policy Architecture

If a grammar is specified as the generator, the device uses a policy grammar that defines the syntax of the policies that can be generated locally. The grammar is used to generate expressions that are defined over an alphabet that includes the local attributes of the various devices described in the interaction graph. The device can select any subset of all of these expressions as its applicable policies. When the generator is a template, place-holders in the template are replaced with the attributes of the discovered devices.

In order to use the generative policy architecture to enable edge sites to develop their own policies, we need to define the interaction grammar and the policy grammars/templates of policies that can be used in the different environment. In this section, we propose a possible interaction graph and a set of grammars that can be used to generate the required policies.

The interaction graph needed for specifying policies in the environment is shown in Figure 10. It consists of three physical devices, the client, the edge site and the cloud site. Attributes like address provided by these physical devices allow them to discover each other. There are two logical devices which are also part of the interaction graph, the application domain and the model. The model is related to an application domain. These logical devices represent the type of information that would be stored and represented at the edge site and the cloud sites.

From the perspective of the edge site, the list of application domains and models that it needs to support can provide it with the input required to generate the policies to work with its system. It discovers new types of domains (which can be available at a discovery URL of the cloud site) and can retrieve the models corresponding to that domain. The edge site can use that to support more than one model or application model as needed.

The policy generators are grammars that are used to specify different types of policies. The types of policies that the edge site may want to generate are described in the section on each type of caching model. A grammar will be used to define the restrictions on how the policy for each type would be generated according to the scheme described in reference [7].

V. CONCLUSIONS

In this paper, we have defined a vision for implementing distributed AI solutions using a model caching architecture. In this architecture, policies are used to define edge sites which can determine which of many different caching modes it should use to perform its functions. We believe that the design should enable the implementations of efficient caches for AI decision making that can be useful in bandwidth constrained environments.

Our next step in this research is to implement a prototype of the edge caching system, and explore how much autonomy we can provide with it using the ideas of generative policies. We would also evaluate the performance of the different types of caching approaches in terms of hit ratios, latency reduction and other performance metrics in different contexts.

REFERENCES

[1] J. Schmidhuber, "Deep learning in neural networks: An overview." Neural networks, vol 61, pp.85-117, Jan 2015.

[2] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):14–22, 2012.

[3] Hermans, Michiel, and Benjamin Schrauwen. "Training and analysing deep recurrent neural networks." In Advances in Neural Information Processing Systems, pp. 190-198. 2013.

[4] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[5] E. Snell, “Healthcare Cloud Adoption Slow Due to HIPAA, Survey Finds”, Health IT Cloud News, Dec. 16, 2015, https://healthitsecurity.com/news/healthcare-cloud-adoption-slow-due-to-hipaa-survey-finds

[6] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises", Business Horizons, vol 58, no. 4, pp. 431-440, August 2015.

[7] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and word2vec for text classification with semantic features". Proc. IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing, pp. 136-140, July 2015.

[8] D. Verma, S. Calo, S. Chakraborty, E. Bertino, C. Williams, J. Tucker and B. Rivera, Generative Policy Model for Autonomic Management, Proc. IEEE Smart World Congress - International Workshop on Distributed Analytics Infra Structure and Algorithms for Multi-Organization Federations, San Francisco, CA, August 2017.