Sommelier: Curating DNN Models for the Masses

Peizhen Guo
Yale University
peizhen.guo@yale.edu

Bo Hu
Yale University
b.hu@yale.edu

Wenjun Hu
Yale University
wenjun.hu@yale.edu

Abstract

Deep learning model repositories are indispensable in machine learning ecosystems today to facilitate model reuse. However, existing model repositories provide a bare-bone interface for model retrieval. The onus is on the user to profile and select from potentially hundreds of choices, barely relieving an average user of the expertise required to design the model in the first place.

In this paper, we present *Sommelier*, an indexing and query system above typical DNN model repositories to interface directly with inference serving or other use cases. Given a desirable accuracy target and resource budget for an inference task category, *Sommelier* automatically searches through the repository for the most suitable model, without requiring manual profiling from the user. Motivated by manual iterative model search processes and typical model design strategies that generate model variants or models with common segments, *Sommelier* organizes DNN models based on their semantic correlation, defined as the probability of models producing the same results. This is further combined with a resource index based on relative resource consumption. *Sommelier* is implemented as a standalone query engine that can interface with an existing repository such as TF-Hub. A case study of 163 models in TF-Hub highlights the extent of model correlation across different model series, suggesting the best candidate model can easily evade manual profiling. Extensive evaluation shows that *Sommelier* returns the ideal model for over 95% of the queries; When interfaced with an inference server, *Sommelier* can reduce the 90th percentile tail latency of inference tasks by a factor of 6 via automatic model switching, far more than typical scale-out system optimizations.

CCS Concepts

• **Information systems** → **Data model extensions**; **Query languages for non-relational engines**; • **Computing methodologies** → **Neural networks**.

Keywords

Deep neural network query engine, deep learning model repositories, semantic indexing

ACM Reference Format:
Peizhen Guo, Bo Hu, and Wenjun Hu. 2022. *Sommelier*: Curating DNN Models for the Masses. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3526173

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3526173

1 Introduction

Deep learning (DL) inference accounts for the explosive growth of analytics workload volume everywhere, in the cloud and on edge devices. Computer vision (CV) and natural language processing (NLP) tasks are the dominant deep learning workloads currently deployed. Meta (then Facebook) reported [55] that the volume of their workloads tripled within two years. These workloads are resource intensive but also increasingly user facing, hence subject to stringent latency requirements.

As it requires significant expertise and computation resources to design deep neural network (DNN) models, it is increasingly common to use a pre-trained model (e.g., ResNet [34] for image recognition), either verbatim or as the basis to *transfer* the model to the target application (e.g., object detection and semantic segmentation). Typical inference serving frameworks provide an API to load an existing model from a given repository. Model designers often start with an existing model and then adapt or retrain it to generate new models. Model testers use similar models to identify adversarial inputs that lie at the decision boundaries. (Section 2) As a result, DNN model repositories have become essential players in existing deep learning ecosystems, e.g., TF-Hub [6] for TensorFlow, PyTorch Hub [4] for PyTorch [56] and Model Zoo for MXNet [14]. These are even more helpful for the general public.

However, existing model repositories provide a bare-bone interface for the user to retrieve a specific model. The onus is on the user to profile and identify precisely which model to use from potentially hundreds of DNNs, including the specific version for a particular DNN design. This level of repository support barely relieves an average user of the expertise required to design the model in the first place. It is especially cumbersome for an application developer who simply wishes to deploy an existing model [75]. Empirical manual profiling is neither scalable or robust, also due to the risk of overfitting to test datasets. Manual model search further inhibits effective run-time adaptation to fluctuating resource availability during prediction serving. A suboptimal model could miss the achievable accuracy target by 10% or waste 20× more resources [4, 30, 70]. Section 2.1 discusses these issues further.

The problem is poised to worsen as new model variants are derived. Partly driven by the growing interest in on-device inference for edge devices and recent work generating a large suite of models customized to heterogeneous settings [12, 30, 72], it is increasingly unscalable to manually select the best fit model.

In this paper, we propose *Sommelier*, an indexing and query system over typical DNN model repositories to automate model selection based on a desirable inference accuracy target and resource budget. Recognizing the difficulty to quantify the exact semantics of each DNN model, *Sommelier* defines a notion of *generalized functional equivalence* between models (Section 3). Strictly speaking, this metric describes an approximation to capture the *semantic similarity* between two models. We use the term "equivalence" to

highlight the *interchangeability* between models in practice. We then formulate the query goal as finding a model most interchangeable with a well-known reference model (e.g., ResNet). This query formulation is motivated by anecdotal evidence of current practice. For example, at Meta (Facebook) and elsewhere, if an engineer wants to add some DL functionality to a product, they would consult a machine learning (ML) expert based on the functionality and performance requirements; The expert will inform the engineer which existing tools or models to try first, and how to iteratively refine the model selection. *Sommelier* is designed to automatically meet such real-world model selection requirements. It also matches common model design strategies such as transfer learning and neural architecture search, which generate models that may be variants of one another or share common substructures. Indeed, our empirical analysis of 120 popular models in existing repositories show over 50 models are derived from the same base model (Section 3.2). For the resource specification in the query, the user simply specifies the budget as using a desirable fraction of the resource footprint of the reference model (Section 5.1). This is motivated by common search criteria for lightweight models [12, 18, 70].

To achieve low-latency query performance, *Sommelier* measures and organizes DNN models with a *semantic index* based on their semantic similarity, defined as the interchangeability of the models producing the same results (Section 4). Rather than quantifying the equivalence between two models *intensionally*, i.e., by requiring that two models produce the same result on every input empirically, *Sommelier* quantifies the functional equivalence *extensionally*, by measuring model performance on a single validation dataset and accounting for the behavior on other validation datasets by including a term for the generalization error bound [8]. The semantic index is further combined with a *resource profile index* to meet resource constraint preferences (Section 5). Given the query format, we mainly need to determine the relative resource consumption between models. Therefore, the resource profiles are defined by hardware-independent metrics, memory usage and TFLOPS; hardware-dependent metrics, such as latency, can be added and estimated with additional information like device specification. The resource profiles are actually *computational complexity profiles*. TFLOPS captures the time complexity whereas memory usage measures space complexity.

Sommelier acts as an explanation database for DNNs. We believe that *Sommelier* can fundamentally change DL framework designs. It completes the code automation for inference serving run-time adaptation, DNN test case generation, and training new models that are currently hard to achieve with manual model selection. *Sommelier* is implemented as a standalone query engine (Section 6), not tied to any specific deep learning repository or ecosystem. It can interface with diverse model repositories as an intermediary (Figure 1) between the repository and the applications.

As a case study, analysis of 163 models in TF-Hub shows that *Sommelier* uncovers hidden correlation between models across distinct collections, beyond what can be expected from manual documentation and the most intuitive manual profiling strategies. Extensive experiments over around 200 models show that when the model differences are distributed uniformly between 0 and 10% (i.e., percentage inference result difference for the same test dataset), *Sommelier* returns the ideal model for over 95% of the queries; When *Sommelier* is interfaced with an inference server, it can reduce the 90th percentile tail latency of inference tasks by a factor of 6 via automatic model switching. This improvement can be far more significant than applying typical scale-out approaches. It hints at a new use scenario for model architecture adaptation. While model compression is normally intended for resource-constrained edge devices, it can produce compact models suitable for cluster settings at heavy loads. *Sommelier* also significantly lowers the expertise requirements, reducing the time needed to select a model manually by up to a factor of 30 and replacing hundreds of lines of script code with less than 10 lines of *Sommelier* queries. (Section 7)

In summary, this paper makes the following contributions: First, we propose a scalable query formulation for DNN models using quantifiable constraints around relative resource usage and functional equivalence between DNN models. Second, we design an algorithmic primitive and code library to automatically extract functional equivalence across DNN models, especially between model segments. Specifically, we generalize testing-based approaches to be benchmark independent, and then generalize the model similarity analysis to model segments. To our knowledge, no previous work automatically infers DNN sub-structures. Third, we build a query system, *Sommelier*, that interposes an indexing layer between the filesystem storing the models and the inference applications; it abstracts away the filesystem and exposes a query interface instead. This way, we can change the way DL frameworks are designed today by enabling a range of automated adaptation and shifting the system bottleneck away from resource scheduling. Finally, evaluation results highlight the potential of automated model switching thus enabled in delivering significant performance improvements without using extra resources.

2 The Need and Gap for DNN repositories

Building DNN from scratch is too expensive. The efficacy of deep learning rests on the quality of the DNN model, but training a sophisticated DNN from scratch is an immense undertaking. This demands comprehensive understanding of optimization theory and neural network internals, enormous amounts of training data, and computation resources. For example, training a ResNet50 network involves carefully choosing optimizers and hyper-parameters, and writing hundreds to thousands of lines of code spanning Python, C++, and specialized libraries (e.g., CUDA, MKL [1]) to deploy the whole training pipeline; the entire training run could take 14 GPU-days, processing nearly 1 TB of training data [34].

Given the colossal cost and steep learning curve of training new models, repositories of DNN models are increasingly adopted in DL ecosystems, to store pre-trained models for diverse model reuse possibilities. For instance, we analyzed around 150 active DL projects on GitHub. Over 94% of them involve building and training upon existing models loaded from some repository. Further, a small set of six common neural network models is chosen by over 60 projects. This confirms the importance of model repositories.

Usage of model repository. Model users can directly choose a pre-trained model from the repository with the required functionality (e.g., object detection) or certain model segments (e.g., visual feature extractors) to build their learning based applications without domain knowledge of DNNs [3, 33].

(i) *Inference serving.* Various deep learning inference serving frameworks (e.g., TensorFlow-serving [54] and Clipper [16]) have been developed to provide run-time support for low-latency, accurate, and robust DNN-based prediction tasks. These systems interpose between end-user applications and the deep learning engines, hiding the complexity of deploying end-to-end DL logic. Applications simply specify the model and provide input data. The inference serving system, integrated with a model repository, will load the specified model, and execute inference tasks with various optimizations (e.g., model freezing [45], multi-tenant sharing [37], multi-model collaboration [78], layer-wise caching [46]).

(ii) *Model design.* The emergence of training techniques such as transfer learning [77] and knowledge distillation [35] facilitates *incremental* new model design by copying (a segment of) an existing well-trained network as the basis and adapting that towards new use cases with substantially less effort (in model designing time and training data size). For instance, popular well-trained neural networks (e.g., ResNet [34] and BERT [21]) are widely utilized for diverse downstream CV and NLP tasks [62, 81], accelerating the training time from *days* to *minutes* [21, 55].

(iii) *DNN testing.* DNN models deployed in safety-critical applications such as autonomous driving need to be robust against adversarial input data, namely specific input values that could trigger abnormal inference results and dangerous behavior, e.g., misclassifying a stop sign. Key to the model verification process is to identify corner case input data. These are typically found by loading a few similar DNN models from the model repositories [41, 57] and exploring the intersection of their decision boundaries.

2.1 Limitations of existing model repositories

Existing model repositories act as a remote filesystem only, with primitive APIs to publish and load a model. To retrieve a model, a user has to specify the precise URL to the model file. This requires significant user sophistication regarding the right model choice.

Inference serving. The next generation of DL inference serving systems are expected to hide complexity; Developer interactions with the associated model repository should only include *high-level specifications* of DNN models (e.g., accuracy, inference latency, and resource usage) as the inputs, rather than the exact model name and version [75]. This is even more so for anyone not familiar with detailed DNN model profiles. From the perspective of a DL inference serving system, the *runtime execution environment* (e.g., queue length, caching strategy) and *application performance goals* (e.g., critical vs. non-critical period) fluctuate [13, 39], making manual and static model selection a poor match for myriad runtime optimization needs and necessitating automatic suggestions.

Model design. Even for domain experts, appreciating the accuracy and resource usage of all models in advance is impossible. For instance, ResNeXt101 [74] and MobileNet [36], two models trained with the same ImageNet [20] dataset for classification, differ by 10% in accuracy and 20× in memory footprint [4, 70]. Further, such numbers are measured under a specific setting only, and could vary with the datasets and hardware platforms. Enumerating models by name and exhaustively profiling each until the best fit is extremely unscalable and not robust against overfitting to the test data.

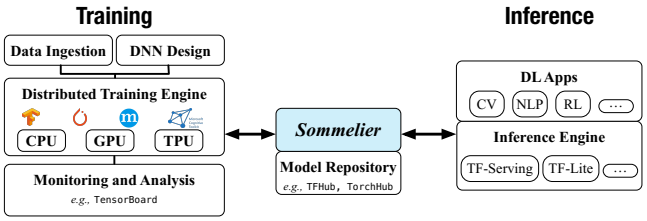


Figure 1: *Sommelier* fills in a gap in the existing DL ecosystem by masking the complexity in DNN model selection.

DNN testing. Since an important model testing step is to find "tricky" input data using similar but not identical models, the quality of model selection dictates the coverage and soundness of the testing process [50, 79]. Ideally, the repository should automatically identify models that exhibit sufficient local differences to explore adversarial examples. Instead, this is currently done manually, potentially testing the same model multiple times unnecessarily.

Summary. All three use scenarios point to the same fundamental limitation of existing model repositories: There is ***no query support*** for the model repository, only a bare-bone filesystem. This then leaves model selection to manual and empirical operations, which is time-consuming and often results in suboptimal decisions. Manual selection effectively precludes run-time model switching when serving inference tasks. Only DL experts with substantial knowledge of DNNs can take full advantage of such model repositories when facing numerous models with diverse profiles [11, 61].

2.2 Requirements for DNN query support

To address the shortcomings of existing model repositories, we build *Sommelier* to provide model query support (Figure 1) to bridge the currently separated phases of DNN model training and inference serving. We next discuss the requirements and challenges.

Canonical model lookup requirements. Fundamentally, *Sommelier* supports a query operation, `query()`, where the method signature captures the user requirements. An example query might take the format of "find a model for vision on embedded devices that uses 80% less memory but only allows within 5% accuracy loss from ResNet" [30, 36]. For a DNN, its inference accuracy and resource usage profile are the key factors forming the multi-dimensional decision space that concerns a user [11, 30, 61]. Note that both are often expressed in *relative* terms against a well-known reference, since it is generally desirable to find the most lightweight model. Therefore, a DNN model query should essentially specify two lookup conditions, ideally in *relative terms*: (i) *model semantics* (e.g., "within 5% accuracy loss to ResNet" or "object recognition with over 95% accuracy over 1000-class ImageNet syntax"), and (ii) *resource consumption* (e.g., "20% of ResNet memory consumption" or "less than 1 GB memory and 0.5 TFLOP"). To fulfill such query requests over neural network models, we need to organize and search for models efficiently, which poses several challenges.

Challenge 1: Characterizing general DNN model semantics. While organizing the DNN models along the resource consumption axis is intuitive, it is not straightforward along the model semantics axis (e.g., by an accuracy target for a specific functionality). The latter relies on a measurable definition of DNN semantics and the right primitive to compare and rank DNNs, neither of which is obvious.

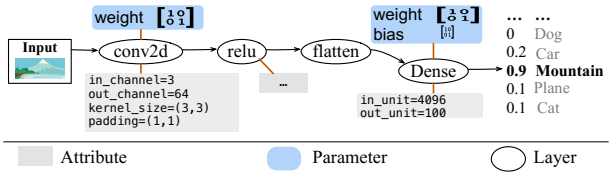


Figure 2: Anatomy of a DNN based inference task.

DNN models are described by directed graphs of mathematical operators, and hence have unique mathematical expressions. However, we observe that using these expressions to define and "compare" DNN semantics does not suit practical scenarios. Instead, given the nature of DNNs and their interaction with existing repositories, we find that assessing and exposing pair-wise semantic correlation between models is more insightful than attempting to quantify the model semantics in absolute terms.

Challenge 2: Quantifying semantic relations between DNNs. Given the primitive to characterize DNN semantics, the next challenge is designing algorithms to measure the semantic similarity between models. First, the nature of such similarity differs depending on whether it is between two DNN models holistically or between model segments. Second, DNN models employ diverse structures and operators, which generic and scalable techniques for model traversal. Third, DNN structures are increasingly complex, hence requiring extensible algorithmic designs. Section 4 describes our approach to these issues. In particular, we build on the theory of generalization bounds to guard against the risk of overfitting test data in empirical assessment. Although it might appear pair-wise correlation could be captured by model designer annotations, in practice, documentation only works for internal repositories, lacks standardization, and limits its understanding to the same family of models (further discussed in Section 7.3).

Challenge 3: Query specification and processing. From a system perspective, the challenge is two-fold: (i) how to design an expressive query interface and specification to cover all user requirements; and (ii) how to design index structures and process the query accurately and efficiently. These are explained in Section 5.

3 Characterizing DNN Semantics

Figure 2 shows an example recognition task using a DNN. A DNN is typically expressed as a directed acyclic graph (DAG), following a dataflow model. Each node in the DAG is a base layer in a network, considered as an atomic unit carrying out a certain operation (e.g., 2D convolution) on its input. Each DNN layer is characterized by *attributes* and *parameters*. Attributes (the grey boxes in Figure 2) describe the types and shapes of the input/output tensors and their dependency. Parameters (the blue boxes) capture the internal states of a layer (e.g., the weight and bias tensor of a `Dense` layer).

3.1 The futility of the conventional view

A DNN is simply a sequence of primitive mathematical operators. Therefore, it is intuitive, and a common practice, to use the mathematical expression of a DNN to denote its task semantics; The difference between model semantics would then appear to be captured by that between their mathematical expressions. However, this common practice is problematic.

Reason 1: Lack of a unique formal representation of the DNN or the DL task. Training a DNN is theoretically understood as a function approximation process, where the exact "functionality" is unknown, but is gradually *approximated* through a finite set of input data and output labels. The training process may impart inherent bias to the training data. The function thus derived is evaluated by how well it *generalizes*, i.e., how accurately it performs the inference task when fed with unseen test inputs. This implies that the same "function semantics" can be "described" via totally different mathematical representations. In formal verification terms, the strongest postcondition for a DL task is not unique because multiple outcomes can be *acceptable* given the same input [26, 42, 68]. This departs substantially from the traditional sense of deterministic program semantics.

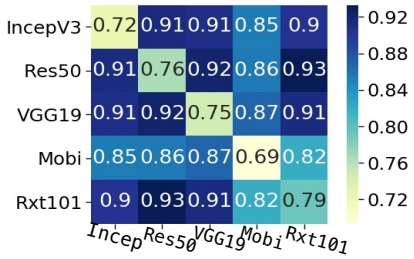


Figure 3: Extent of equivalence between DNN models.

Reason 2: The correctness of a DNN is *tunable*. Traditional program correctness is a binary property ("yes" or "no"), but it is statistical for DNNs. It is common to revise the DNN structure to adjust the tradeoffs between accuracy and other performance metrics, but this does not change the functional semantics of the task. For instance, neural architecture search algorithms (e.g., OFA [12] and MnasNet [69]) adjust the neural network structural complexity factor, between 0.1× to 10× in exchange for the right accuracy targets that vary by almost 10%, in order to balance between acceptable performance and resource footprint on edge devices. Minerva [60] prunes computation on the hardware to achieve 3× processing speedup on edge devices, at the expense of 2 to 10% accuracy drop.

3.2 Empirical analysis

Where distinct models agree. We next empirically show the discrepancy between the *mathematical difference* and *functional equivalence* between various DNNs. We select 5 widely used DNN models (Resnet50, Inception, ResNext101, VGG19 and MobileNet), all pre-trained with ImageNet for image classification, and feed the same test input to all of them. In Figure 3, the off-diagonal entries show the fraction of results (corresponding to the top-1 accuracy) that agree completely, while the diagonal entries show the inherent top-1 accuracy for each model. Interestingly, the output agreement ratio *between* models is significantly higher than their inherent accuracy values. This implies that these models are functionally equivalent and highly interchangeable in practice, yet none of the models is the definitive one for the image classification task.

This hints at the futility of attempting to characterize the *absolute* semantics of individual models with mathematical expressions. Indeed, recent DNN verification efforts [26, 42, 68] demonstrate that it is impractical to generate specifications to describe *holistic* (rather than *local*) properties of a DNN model.

Model correlation in common repositories. We next examine 120 popular DNN models, each present in all three common repositories, for TensorFlow, PyTorch, and GluonCV. Each model is trained

with one of only 4 distinct datasets. Further, a common neural network structure (e.g., ResNet block) can be transferred from its original model (ResNet) to over 50 different DNN models spanning over tens of different domain-specific tasks [6, 17, 62, 81]. Since a DNN extracts features from the input data to make inference decisions, the *common* training data and network structures generate implicit correlation between feature extraction in *distinct* DNNs.

Observations. The first experiment confirms mathematical difference is inadequate to describe similar DNN functional semantics, whereas the second analysis suggests that such model correlation is widespread due to the typical training processes today. Therefore, any model selection strategy needs to explicitly take both into consideration. Meanwhile, the inherent correlation between the features identified by different models is more deterministic than the individual feature extraction processes (i.e., distinct DNNs), which sheds light on an alternate view of DNN model semantics.

3.3 Alternate view: Model equivalence

Inspired by the observations above, we define DNN semantics by instead exploring the correlation between models. Users typically know the accuracy and resource profiles of some well-published models, e.g., ResNet for computer vision and BERT for NLP. Therefore, we can assess the semantics of a model with respect to a well-known reference, supplied by the user or defined by default.

Functional equivalence between DNNs. We formally define the *functional equivalence* between DNNs as ***the interchangeability of the models to achieve the inference task.*** Given a model M_1 and an arbitrary dataset D_1 containing input data and the ground-truth results of the inference task. A second model M_2 is (approximately) functionally equivalent to M_1 iff feeding D_1 to M_2 achieves a quality of result (e.g., 95% classification accuracy) comparable to M_1's, up to a user-specified difference threshold ϵ (e.g., 5%).

The rationale is that (i) the equivalence measurement between DNNs is decoupled from their concrete mathematical representations; and (ii) the *threshold* is a control knob for users to customize the level of relaxation acceptable to suit their unique needs. Note that, in machine learning theory, while many terms can be used to express the *similarity* between two functions, these do not translate to the *interchangeability* between (sub-)models. Therefore, the above definition is needed for our particular consideration. This notion of functional equivalence can be applied beyond deep learning, but we restrict our discussion to DNNs in this paper.

4 Identifying DNN Equivalence

Having transformed the problem of unambiguously specifying the semantics of a DNN to assessing the functional equivalence between models, we next develop algorithms for this equivalence assessment. Given the type of model variants commonly seen today, there are two cases to consider. First, two DNNs might be designed differently but trained with similar data to achieve the same task (e.g., recognizing animals). In particular, one model could be a structurally more compact version of the other. The two models thus exhibit functional equivalence holistically. Second, model variants are frequently derived from a common model base, but *transferred* and *fine-tuned* to different downstream tasks (e.g., emotion detection and question answering), then the functional equivalence relation exists between the common base segments of two models.

Therefore, we describe how to quantify the equivalence *extensionally* (i) between holistic DNNs, and more importantly (ii) between common segments of DNN models, in both cases leveraging the generalization theory of DNNs to lower-bound the equivalence measure in a *dataset-independent* fashion.

4.1 Equivalence between whole models

We can simply treat each DNN as a black box and compare models as a whole in three steps. We first check the "structures" of the input and the output, i.e., the data types and shapes of the tensors, to quickly filter out completely different models (e.g., for tasks that are not comparable), and then use a validation dataset to derive an empirical *quality of result* (QoR, e.g., accuracy, mAP, mIoU, etc.) difference between models. These two steps resemble a static analysis of type-check then value-check in a compiler optimization process. This *dataset-dependent* empirical difference is **refined with a generalization error bound analysis** (details in [8]) to become *a dataset-independent QoR difference bound*, then compared against the acceptable difference threshold ϵ to determine model equivalence. This generalization analysis is essential to make our approach *extensional* instead of *intensional* (Figure 11).

Input and output layer check. We check the input tensor shapes of the candidate models to determine if they possibly capture the same semantics. However, this can be misleading because resizing and other preprocessing can be applied to the same raw input source to manifest in different shapes. To cope with this, the model designers can specify in the configuration how to preprocess inputs, as well as register custom preprocessors. The strict comparison between input shapes is invoked only in the absence of preprocessing.

The model outputs are typically derived in three ways based on the task category, *classification* (semantics defined by the highest-valued dimension of the output vector), *regression* (semantics defined by the whole output vector), or a combination of the two when there are multiple outputs. For regression-style outputs (e.g., for object detection, word embedding), we simply observe the output shapes. If the shapes are identical, we pass the two models to the next checking phase. For classification-style outputs (e.g., for object recognition), a finer-grained check can be additionally conducted if the output syntax is specified, namely the syntax label of each output dimension (e.g., dimension *i* maps to *dog*, dimension *j* to *cat*, ...). Such an analysis can eliminate models with the same output shape but carrying different syntax.

Assessing functional equivalence. We next feed the validation dataset to two candidate models and measure the average QoR difference. Typically, the QoR goal is just the optimization objective for model training. Otherwise, we compute the l_2 distance between the outputs from the two models on the same input, then average this distance over the dataset as the default QoR difference.

So far, we have an empirically measured QoR difference which might be specific only to this validation dataset. In order to generalize the empirical assessment without exhaustively testing *all* datasets, we leverage the generalization theory of DNNs [10, 53] to upper-bound the QoR difference (or equivalently, to lower-bound the semantic similarity between models) *independent of the validation dataset selection.* In brief, we add to the empirical QoR difference a *generalization error bound* derived from the architecture of the

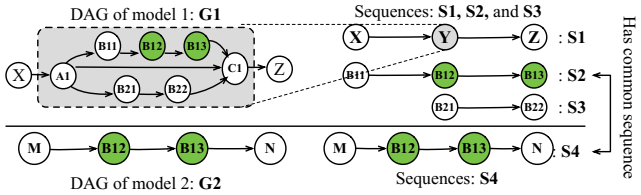


Figure 4: Extracting model segments recursively.

DNN. The generalization bound is expressed as:

$$\tilde{O}\{(\frac{1}{\gamma^2 n} d^2 max\|f(x)\|_2 \sum_{i=1}^{d} \frac{1}{\mu_i^2 \mu_{i\rightarrow}^2})^{1/2}\},$$

where γ is determined by the accuracy metric definition of the inference task, n is the size of the validation dataset, d denotes the total number of layers of the DNN, $\|f(x)\|_2$ denotes the output vector's l_2 norm, and μ_i and $\mu_{i\rightarrow}$ are inter-layer factors calculated from the weight matrices of the adjacent layers (details in [8]).

Finally, we determine the two models to be functionally equivalent if the upper-bound QoR difference is smaller than a pre-defined default or user-specified threshold ϵ.

4.2 Equivalence between model segments

For DNNs generated via transfer learning or model adaptation, two DNNs may not be equivalent in their entirety, but share functionally equivalent segments (e.g., a stack of layers). Therefore, we revise the equivalence definition and analysis for DNN model internals. There are two main challenges: (i) extracting structurally identical model segments, and (ii) assessing the functional equivalence of them. Our detection algorithm proceeds in two steps accordingly. For (i), we adopt a lightweight approach based on common DNN structures. For (ii), our key insight is to estimate layer-wise output difference inductively between model segments based on the nature of error propagation over a DNN.

Revised equivalence definition. For model segments, we first need to quantify their roles in the whole model, since the model-wide validation datasets and quality of result (QoR) metrics are not directly applicable to intermediate segments. Suppose we have a segment S from model M, and another segment S' structurally identical to S. We can derive a *twin* model M' from M by replacing the segment S with S'. Now, we can translate the functional equivalence between S' and S to the equivalence between M' and M, as defined previously (Section 3.3).

Extracting model segments. Unlike the whole model scenario, only checking the first and last layers around intermediate model segments is far less informative for filtering out unrelated models. Instead, we view the DNNs as DAGs and extract the common sub-graphs as the candidates that are possibly equivalent. However, optimally detecting common sub-graphs between two graphs is well known as an NP-hard problem and is simply not tractable. Fortunately, unlike general graphs with arbitrary node connectivity, DNNs tend to connect layers sequentially, and only involve a small set of parallel branches locally (e.g., residual connections in ResNet [34]). Therefore, our algorithm instead finds the longest common *operational sequence* as the candidate segments, which reduces the complexity to $O(N^2)$, where N is the number of layers.

We first recursively find the longest operator sequence from each DAG. For example (Figure 4), the sequence $S1$ is extracted first; then, zooming into the "operator" Y, another two sequences of operators, $S2$ and $S3$, are further extracted from the graph $G1$. Now, given a set of sequences (i.e., $S1$ to $S3$) from each DAG, we find the longest common sequences (green shaded) between the two sets as the candidate model segments for further assessment.

Layer-wise output difference estimate. We estimate the upper-bound output difference between two model segments *layer-wise from the input layer to the output layer inductively*. For the *base case*, we can show how the input difference (at layer 0) is transformed by any operator in the first layer of the segment to the *first-layer output difference* (i.e., also the *input difference to the next layer*, effectively an error term). For the *inductive step*, the output difference bound for layer i is calculated from the theoretic upper bounds for layer $i-1$ following the per-layer input and operator constraints. At layer i, any linear operator adds to or multiplies the errors from layer $i-1$ by a scalar, and non-linear operators upperbound the errors following the operator definitions. Then, by induction, the difference vector between the output values captures the deviation between the two model segments over all common layers. We next analyze the per-operator effect on error propagation.

We classify DNN layers (operators) into three categories: *linear* operators, *non-linear* operators, and *multi-source* combinations. Linear operators essentially cover all kinds of layers invoking matrix multiplication (`FullyConnected`, `Convolution`, `Embedding`, etc.). Non-linear operators include activation (`ReLU`, `tanh`, `sigmoid`, etc.), pooling (`maxpooling`, `meanpooling`, etc.), and normalization. Multi-source combination refers to merging multiple input sources into a single output (`add`, `multiply`, `concat`, etc.). Note that even though recurrent operators (RNN, GRU, LSTM, etc.) are typically viewed as distinct operators, their essential computing logic is no different than a combination of the aforementioned basic operators. Therefore, each recurrent operator itself can be treated as a model segment for error analysis.

For the *linear* operators (the computation kernel of almost all DNN layers), the output difference of the current layers (e.g., $B13$ in Figure 4) between segments comes from two sources: (i) the output difference of the previous layers ($B12$ and earlier) propagated to the current layers; and (ii) the additional output difference incurred by the weight parameter differences between the current layers ($B13$). Suppose W and W' are the weight matrices (with identical sizes) of two counterpart layers in segments S and S', and ΔX is the upper bound of the difference vector. At layer i, we denote the input and output difference vectors with respect to S by ΔX_S^i and ΔX_S^{i+1}, and the weight matrix difference by $\Delta W_S = W' - W$. Clearly, we have $\Delta X_S^{i+1} = W \cdot \Delta X^i + (\Delta W) \cdot X^i$. Now, we can derive the per-layer upper bound $max\|\cdot\|$ as

$$max\|\Delta X_S^{i+1}\| \leq \lambda_{max}(W) \cdot max\|\Delta X_S^i\| + \lambda_{max}(\Delta W_S) \cdot max\|X^i\|,$$

where $\lambda_{max}(W)$ denotes the largest singular value of matrix W, and $max\|X^{i+1}\| = \lambda_{max}(W) \cdot max\|X^i\|$. The largest singular value of a matrix indicates the largest scaling effect among the dimensions acted on by the matrix. Each $\lambda_{max} \cdot max\|\cdot\|$ term thus maximally scales the previous error bound, and the final addition combines both bounds. Note that, for `Convolution` layers, the kernels are always internally reshaped into a single 2D matrix before calculating output. Therefore, they are treated in the same way as `FullyConnected` layers even though the latter often involve multiple kernels on multi-dimensional inputs.

Next, we derive the output difference bounds for *non-linear* operators. Consider *activation* layers (`RELU`, `tanh`, and `sigmoid`) first. All these activation layers and their variants (e.g., `LeakyReLU`) follow $|activation(x)| < |x|$, which means the input difference bound itself could serve as the upper bound of the output difference. Then, for *pooling* layers, it is easily proved that the l_2 difference of the outputs is always smaller than or equal to that of the input difference. For *normalization* layers, the output difference is scaled by a factor determined by the length of the original output vectors. Thus, we can simply derive the upper bound as $\Delta X^{i+1} = \|\Delta X^i\|/\|X^i\|$.

Finally, we analyze *multi-source* combination operators. Intuitively, we treat the difference vectors of each input source as an independent random variable. Applying the relevant mathematical expressions to each, as outlined above, then generates the statistics of each field in the output difference vector.

To sum up, for each type of popular neural network layer (operator), we propose an approach to derive the output difference upper bound from its input difference and the weight matrix of its possibly equivalent counterpart. This completes the inductive step.

Completing equivalence assessment. Say we have identified equivalent segments between models M and N and calculated the output difference bound per segment pair, the equivalence assessment between the original M and a maximally replaced M proceeds as follows. The same can be done for N vs. a partially replaced N.

(i) Denote the set of all replaceable segments of M (in common with N) by S_M. We feed random inputs to M and get the outputs from each segment in S_M and for the entire M.

(ii) Next we *estimate* the output of M if all segments in S_M are replaced. Each segment-specific intermediate output is perturbed with Gaussian noise scaled to the corresponding per-segment output difference bound, and fed to the rest of M (after the segment just ran). Adding random noise emulates replacing a segment and incurring maximal result quality degradation. This way, we can calculate the difference between each pair of the segment-replaced and unperturbed outputs of M; Across all inputs, we then obtain the QoR difference.

(iii) If the QoR difference exceeds the threshold ϵ, we gradually remove segments from S_M in the order of increasing computational complexity, and recalculate the new QoR difference via steps (i) and (ii), until it falls within the threshold.

Two points to note. First, since this output difference analysis takes as input the initial test input difference, it intrinsically unifies the *equivalence* notions between pairs of *inputs* [22, 31, 32], *intermediate results* [47], and *models*. Second, adding Gaussian noise (i.e., completely random error) in step (ii) captures the worst case for the equivalence analysis, whose result then serves as a safe bound for arbitrary scenarios. Since the noise and test data distributions vary by model and are usually unknown, assuming less random error inherently biases the results towards certain scenarios (Section 3.1).

4.3 Discussion

Asymmetry of model comparison. Our "functional equivalence" metric is non-symmetric, i.e., the equivalence scores between two models differ based on which model is the reference model. This asymmetry is by design and we believe it better captures real-world scenarios. For example, when a tiny model and a huge model share

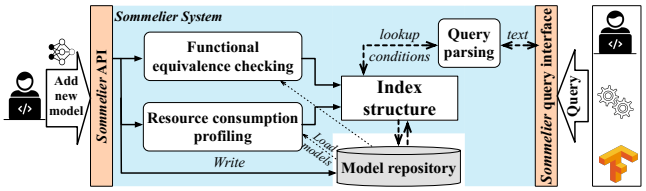


Figure 5: *Sommelier* system architecture.

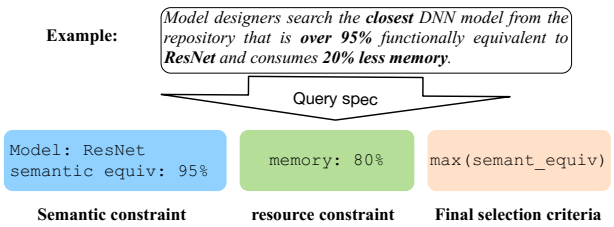


Figure 6: Specifying a concrete use case as a DNN query

a "functionally equivalent" operator sequence, intuitively, replacing the sequence for the tiny model might incur more accuracy degradation than the huge model, because the sequence plays a more impactful role in the final output for the tiny model.

Limitations. There are two main limitations to the above algorithms. First, *Sommelier*'s applicability heavily depends on sufficiently capturing functional equivalence between DNN models and segments, which in turn depends on the theoretical results on generalization bound analysis. They currently work well for most common types of DNN operators, and we expect relevant theory to evolve as new operators emerge. Second, for now our algorithms only capture the models with identical topology. We hope to extend our algorithms to capture broader scenarios in future.

5 DNN Model Query with *Sommelier*

Building on the above analysis of DNN models, we design *Sommelier* (Figure 5) to support DNN model queries. The core system is built on a pair of index structures, a semantic index and a resource profile index, to "rank" DNNs and support range queries.

5.1 Formulating DNN model queries

Recall the model lookup requirements in Section 2.2. Due to the prevalence of DNN tailoring and on-device inference [12, 18, 70], the model search targets are typically described in terms relative to a well-known reference model. We define a query specification to express desirable model performance and resource usage (Figure 7). Such queries are not about exact matches, but like multi-attribute range queries that jointly consider multiple lookup conditions.

A user query specifies the *semantic constraint*, the *resource budget*, and the *final selection criteria*. The semantic constraint is defined by a reference model and the functional equivalence threshold with respect to this reference. Resource constraints include (relative) computational complexity and memory consumption. These are meant to indicate the user's *preference* for resource usage. A user does not need to know the exact resource footprint of any model. Final selection criteria outline any additional method(s) to select the final output among the retrieved candidate models (e.g., the model with the "most similar" function, or select with user-defined utility functions). If the user has no prior knowledge of a suitable reference model, they can specify the inference task category instead and *Sommelier* supplies a default reference model.

Figure 6 shows a query example, where a model designer wishes to find a DNN that is *most* interchangeable with the latest version of `ResNet` (i.e., equivalent to ResNet 95% of the time) but consumes `20%` less memory and takes `40%` less computation time. We use TFLOPS and memory footprint as the two most representative resource metrics to explain the *Sommelier* design, but *Sommelier* is not restricted to these two metrics. Custom metrics (e.g., latency) can be plugged in easily (shown later in Figure 7).

5.2 Semantic index

Sommelier leverages an index structure to track the functional equivalence relations between stored DNNs, so as to process queries efficiently without having to compare the semantics between each pair of DNNs per query. The top-level structure of the index is a hashtable. For each entry in the table, the key is the hash fingerprint of a DNN, and the value is a list of candidate records, each of which consists of a candidate DNN and its functional equivalence score (i.e., the dataset-independent QoR difference bound) to the keyed model. The records within each candidate model list are maintained in a descending order according to the functional equivalence score. This way, the hashtable maintains the mapping between a model to all its functional equivalents.

Insertion to the index. When a new DNN model (M_n) is added, *Sommelier* randomly selects 5 existing models in the repository and conducts a pairwise semantic analysis between M_n and the selected models. The differences between (M_n) and the other models in the repository can be derived transitively: suppose models X and Y differ by A, Y and Z by B, then the semantic difference between X and Z is bounded by $|A - B|$ and $|A + B|$. Empirically, this sampling approach dramatically improves scalability without degrading query quality much. Currently, there is one index for the entire repository, since the majority of existing models are for either CV or NLP tasks. This can be easily extended to one index per inference task category. We assess the functional equivalence between two models regardless of their intended inference tasks.

A new entry (`Rn`) is created in the index table representing M_n. (i) For whole models, suppose an existing model M_1 has functional equivalence level $L_{1 \leftrightarrow n}$ to the new model M_n. Then, the model M_1 along with $L_{1 \leftrightarrow n}$ is added to the candidate list of the entry `Rn`. (ii) For model segments, suppose a segment S_1 of an existing model M_1 has equivalence level $L_{S_1 \rightarrow S_n}$ to a segment S_n of the new model M_n (e.g., interchangeable from S_1 to S_n for M_n). Then, a model M'_n synthesized from M_n by replacing S_n with S_1 is added to the candidate list of entry `Rn` along with the equivalence level $L_{S_1 \rightarrow S_n}$. Separately, for each entry of the existing models, the new model M_n is also added to their candidate list following (i) and (ii).

Lookup with the index. When a query is submitted with the reference model M_n and the functional equivalence threshold as the arguments, *Sommelier* will first locate the key by calculating the fingerprint of the reference model, and then, from the candidate list, collect as the output all the models whose equivalence level exceeds the threshold. An output model M_i can be a real model that is holistically equivalent to the input model M_n. Alternately, M_i can be synthesized by replacing a segment S_n (from input model M_n) with S_j (from a real model M_j) such that S_n and S_j are equivalent.

5.3 Resource profile index

Sommelier builds another index structure to record the resource profile of each DNN model, separately handling relative and absolute resource profiles in a hardware-independent fashion. Each entry of the resource index is a (*key*, *value*) pair, where the key is a vector whose fields correspond to the usage numbers of certain resource types (e.g., (memory, TFLOPS)), and the value is the DNN. *Sommelier* uses Locality Sensitive Hashing (LSH) with a cosine hash family [19] to organize the entries for fast distance-based range search over resource vectors. The optimal LSH parameters vary by scenario but can be empirically set with a few sample entries.

Insertion to the index. The essential step for inserting a new DNN model into the resource index is to generate the resource profile vector. For relative resource constraints, the profile vectors corresponding to all models are normalized (e.g., to a default reference model and hardware platform) to achieve hardware independence. When the reference model is not defined, we leverage static estimation and run-time measurements to generate the resource vectors. For computation complexity and memory footprint, we sum up the TFLOPS and intermediate data sizes of all computation-intensive operators in the model. The latency estimation additionally involves preparing the model runtime, as the execution configurations could affect the numbers [18]. To handle this, *Sommelier* follows the typical practice of separately maintaining a per-operator latency table, which includes the run-time latency of each type of basic neural network operators [58]. Given a new DNN model, its estimated latency is essentially the sum of the individual latency of all operators along the longest sequence between the input and the output. Specifically, for a *sequence* of operators that cannot be parallelized, its latency is calculated by adding up the per-operator latency from the table; For a set of *parallel(izable)* operators (or operator sequences), the latency is estimated based on the path (among all concurrent ones) that takes the longest time to complete.

Lookup with the index. When a query is submitted with the resource constraints, *Sommelier* first converts the resource specification into the constraint vector as mentioned, and then uses the vector to query the LSH-based index. Finally, among the returned models with closest resource profile, those that satisfy the constraints in all dimensions will be the outputs.

5.4 Query processing

A query submitted to *Sommelier* is first parsed into an abstract syntax tree (AST), from which the user-specified query conditions are extracted to formulate three query processing steps. Each step is determined by a filtering constraint, the *semantic constraint*, the *resource consumption constraint*, or the *final selection criteria*.

Based on the two indices, DNN queries submitted to *Sommelier* are handled as pipelines of *filtering* operations. Each query stage takes the information from the corresponding part of the query to configure the filter, executes the filtering logic on the index structure, and then intersects the output models from the current stage with the models from the previous stages. Noticeably, for the resource consumption filter, the filtering condition is further represented as a multi-dimensional vector. For instance, *memory less than 200 MB, computation complexity less than 50 GFLOPS, and latency less than 30 ms* is simply represented as a vector $(200, 50, 30)$.

5.5 Discussion

Configuration knobs. *Sommelier* provides configurable knobs for different scenarios. The generalization bound analysis can be in the *on*, *off*, or *custom* mode. Namely, the user can choose to run analysis on a default dataset, disable the analysis, or specify a custom dataset to run the analysis for specialized scenarios. Further, the semantic and resource indices are configurable, e.g., via the LSH data structure configurations, to balance the computation complexity and indexing accuracy.

Supporting developer annotations. The above description for the index generation assumes no metadata available for any model. If any annotation is available, for example, noting down the model accuracy and resource footprint in a particular setting, *Sommelier* can incorporate and "translate" this information to our standard indexing metrics, in place of the corresponding analysis. However, such annotations are unlikely to replace the built-in analysis provided by *Sommelier* unless they can exhaustively cover all information about the model in any run-time settings.

Resource metrics. To some extent, our resource profiles are actually *computational complexity profiles*. TFLOPS captures the time complexity whereas memory usage captures the space complexity. TFLOPS is widely adopted by most platform-aware DNN adaptation work [15, 27, 72] and independent of the specific types of hardware and frameworks, but with the drawback that it is not always accurate when further translated into platform-specific metrics such as latency. To overcome this limitation, *Sommelier* prepares the inference engine runtime for each new incoming model on locally available hardware platforms (e.g., CPU, GPU, and TPU) and collects the actual performance numbers of the additional metrics (e.g., latency). According to publicly available statistics [33, 38], a small set of common types of platforms could support over 95% of all types of workloads in a large company. This confirms the feasibility of using a small set of hardware in the *Sommelier* runtime to support platform-aware metrics [18].

Persistence. *Sommelier* keeps only the two index structures in memory. All the models are stored in the storage system, and only the storage locations are kept in the indices. As the two indices use vanilla data structures such as hashtables and LSH, both indices are lightweight and can be populated to disk when they grow large.

Framework agnosticism. It is sometimes inevitable to interact with the DL framework runtime to accurately measure the resource consumption and analyze the model semantics. However, neither the model semantics analysis nor the resource profiling differs significantly between TensorFlow, PyTorch, MxNet, and others. Hence, *Sommelier* is not tied to or limited by any specific DL framework.

6 Implementation and Case Studies

We implement *Sommelier* as a standalone query engine taking existing model repositories as its data connectors. The implementation consists of around 6000 lines of C++ and CUDA code for neural network graph and operator definitions, functional equivalence assessment, and query processing, plus around 1000 lines of Python code to import and export DNNs between *Sommelier* and the ONNX format [2], a universal neural network model representation compatible with all mainstream frameworks such as TensorFlow and PyTorch. In particular, the module to assess functional equivalence can be separated out as a standalone tool or common library for model analysis. The source code is available at [5].

```
<query> ::= 'CORR' ref-model
            'ON' { <attr> ('>=' | '<=' | '<' | '>') threshold }+
            'ORDER BY' { <metrics>(<attr>) }*
            'WITH' { exec-spec }*;
<attr> ::= 'SEMANTIC' | 'MEM' | 'CPU' | 'LATENCY';
<metrics> ::= 'MAX' | 'MIN' | UDF;
```

Figure 7: *Sommelier* query syntax.

```
/* Online: inference serving */
while (input_buf.has_next()) {
  // check certain conditions
  ctx = check_execution_environment();
  // Without Sommelier
  // reselect models by fixed conditions
  if (ctx.memory >= 10G && /*...*/)
    model = repo.load("ResNet152");
  else if (ctx.memory >= 5G && /*...*/)
    model = repo.load("ResNet50");
  else if (ctx.memory >= 1G && /*...*/)
    model = repo.load("ResNet18");
  // ... else if ...
  else model = repo.load("MobileNet");
  // With Sommelier
  model = sommelier.query(ref='ResNet50',
    mem=ctx.memquota,flops=ctx.compquota);
  data = input_buf.next();
  res = model.predict(data);
}
```

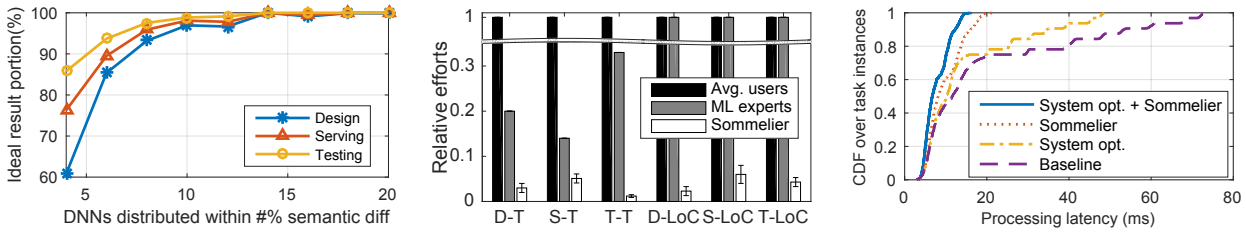
```
/* Offline: DNN query */
// Without Sommelier - exhaustively profile
// specify model performance targets
targets = {...};
// prepare and load dataset
dataset = data.load("Cifar10", "test");
// Exhaustive trying
for(model : repo.models()) {
  accuracy_score = model.validate(dataset);
  // check resource usage
  res_usage = check_resource_usage();
  if (satisfy(accuracy_score, res_usage,
targets))
    return model;
}
// With Sommelier - submit a query string
CORR ResNet50
ON memory < 80% AND flops <= 50%
ORDER BY MAX(SEMANTIC_CORR)
WITH (device = gpu-rtx2080)
```

Figure 8: Pseudo-code for two case studies.

APIs. *Sommelier* connects with a user-specified DNN model repository during initialization. *Sommelier* further exposes a `query()` API in place of the original interfaces between users and the model repository. It takes a query command (syntax shown in Figure 7) as the input and returns a list of selected DNN models, or `null` if none satisfies all the query predicates. The emphasized terms are supplied by users or other DL framework components (e.g., inference serving systems). ref-model is the name (or ID) of a reference model (if left empty, a default model is chosen based on the type of inference task), threshold is the functional equivalence threshold, and, optionally, exec-spec specifies additional execution settings (e.g., hardware information, running mode, and batch size) in key-value pairs to help build DNN resource profiles.

Porting to other DL frameworks. *Sommelier* can easily interface with different DL frameworks. DNN representations are interchangeable between frameworks via ONNX [2]. Further, existing repository APIs are mostly equivalent. For instance, loading models from TF-Hub needs a call of `tfhub.KerasLayer(model_url)`, and from PyTorch Hub, `torch.hub.load(path, name, pretrained)`. Hence, only 3 lines of configuration change is needed to migrate *Sommelier* across model repositories.

Online case study: Inference serving. *Sommelier* supports selecting DNN models with high-level performance goals and resource budgets. This then easily enables automated model switching for DL inference serving (Figure 8, left). Specifically, on receiving an inference task, the inference server first collects the current machine conditions (e.g., task queuing length, allocated GPU memory/shares), and then just queries *Sommelier* (green shaded block). *Sommelier* returns a (set of) DNN model(s) that best fits the current server conditions and allocated resources. These are done automatically by the inference server during the run time. Without *Sommelier*, the developers have to *hardcode* all the model variants and switching conditions, and update the code to incorporate each new scenario and model (gray shaded block).



(a) Query quality (*Sommelier* vs ideal).

(b) Saving in time and manual effort.

(c) Run-time inference latency.

Figure 9: End-to-end performance.

Offline case studies: DNN design and testing. *Sommelier* could fundamentally change the offline usage of the model repositories.The right half of Figure 8 compares the processes needed and the complexity of optimal DNN model selection with and without *Sommelier*. For *DNN design*, using *Sommelier* will accelerate the iteration speed to designing new models by orders of magnitude. Suboptimal model bases are directly skipped by *Sommelier*, without triggering time-consuming training processes. For *DNN testing*, *Sommelier* fills in the last piece of fully automated testing process [41, 57, 71]. When a DNN arrives for robustness testing, the pipeline *automatically* takes the model as the reference and queries *Sommelier* for N functionally equivalent variants. These N variants then form the adversarial input detector for the tested model. Without *Sommelier*, the adversarial input detector is *manually* constructed by the developer for *every* single tested model.

7 Evaluation

The key to *Sommelier*'s performance is to build the semantic and resource indices effectively and efficiently. Therefore, our goals here are to (i) show how the query system can be used for the use cases outlined in Section 2; (ii) evaluate the semantic analysis algorithms in Section 4; (iii) use TF-Hub as a case study to evaluate the index structures and analyze what they reveal about the models; and (iv) evaluate the overhead of various operations.

DNN model benchmarks. *Sommelier* is oblivious to where the model is eventually run. Therefore, we use models of different sizes designed or tailored for diverse execution settings. We prepare two sets of DNN models: (i) a synthetic repository of over 200 models we generate ourselves, transferred from six widely used pre-trained models: three for vision (*image recognition* [34], *object detection* [62], and *semantic segmentation* [81]), and the other three for NLP (*sentiment analysis*, *question and answering (Q & A)*, and *named entity recognition*) [21]. This gives us fine-grained control in terms of different functional equivalence levels to extensively evaluate the algorithms; (ii) We also use over 160 most widely used TF-Hub models from the top 30 most recommended collections (covering CV, NLP, and other tasks) to show how *Sommelier* indices perform in realistic settings (Section 7.3).

Datasets. We use a few widely-used datasets to tune, validate, and assess functional equivalence between models: ImageNet [20], Caltech256 [28], and SUN397 [73] for *object* and *scene recognition*; PascalVOC [24] and MSCOCO [49] are used to fine-tune *object detection*; Ade20k [82] to fine-tune *segmentation*; SQuAD1.1 [59], IMDB [51], and CoNLL03 [66] to fine-tune *Q & A*, *sentiment analysis*, and *named entity recognition* workloads respectively.

Hardware. Unless otherwise noted, we use a single Linux server with a quad-core 2.3 GHz Intel Xeon CPU, 64 GB memory, and an NVIDIA RTX2070 GPU. This covers inference scenarios broadly, since they are run on a single server whether at the edge or in the cloud. *Sommelier* applies to both.

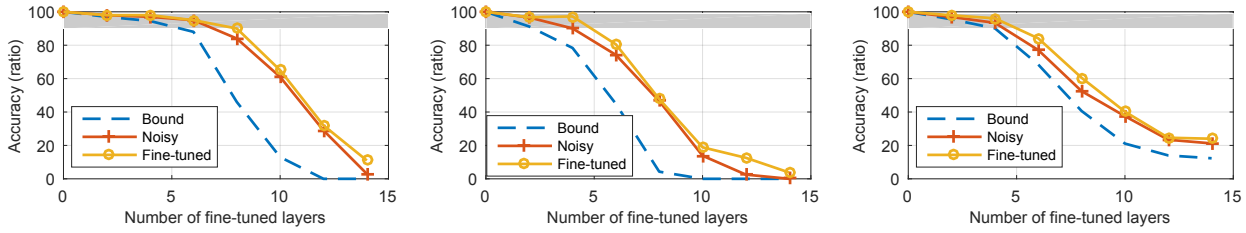
7.1 End-to-end performance

Settings. Following the setups above, we evaluate *Sommelier* in an end-to-end fashion in the context of the three motivating examples and case studies (Sections 2 and 6), i.e., model design, model testing, and inference serving. All experiments use the real-world datasets and models mentioned above. These cover both whole model similarity and model segment similarity. *Model testing* also borrows the model settings specified in DeepXplore [57].

Performance metrics. We evaluate *Sommelier* in terms of *query quality*, *time and manual effort*, and *inference tail latency*. The first two capture how well *Sommelier* selects DNN models (measured as the portion of DNN models selected by *Sommelier* being ideal) and how it simplifies interacting with the DNN model repositories (measured in the time and lines of code needed), respectively. For online use scenarios such as inference serving, we further evaluate the *tail latency* of the inference tasks with and without *Sommelier* to examine the usefulness of automated DNN selection.

Query quality. Figure 9(a) shows the portion of query output models matching the ideal model. When model differences are distributed evenly between 0% to 10% (i.e., returning different inference results for up to 10% of test input), *Sommelier* returns the ideal model for over 95% of the cases. Even when all models are functionally different from one other by at most 4% (the most extreme case where all models are "usable"), *Sommelier* still consistently returns the ideal one for over 60% of the cases. In such cases, the candidate models are already nearly identical. Without additional constraints, the choice between them is essentially random. We do not observe any significant inference quality drop from suboptimal choices.

Time and manual effort. We conduct the experiment with 10 ML experts (e.g., PhD students focusing on DL) and average users (e.g., sophomores without extensive experience with DL). We explain to them the goals of the three tasks and where to find the available models to achieve the tasks. Figure 9(b) shows the relative time and lines of code needed for manually profiling and selecting models, compared to using *Sommelier*. The prefixes "D-", "T-", "S-" stand for "design", "test" and "inference serving" respectively, while the suffixes "-T" (the leftmost three groups of bars) and "-LoC" (the rightmost three groups of bars) indicate respectively the profiling time and lines of code needed. *Sommelier* reduces the profiling time



(a) Object/Scene recognition. (b) Object detection. (c) Image segmentation.

Figure 10: QoR difference bound and actual accuracy loss given varying fine-tuning levels and datasets for three vision tasks.

needed by up to 30×, and replaces hundreds of lines of script code with 10 lines of queries.

Inference accuracy and tail latency. Finally, we highlight how *Sommelier* can help with reducing the tail latency of inference serving via automated model switching. Our baseline is an inference model server interfacing with a vanilla model repository, where the application developer manually specifies a fixed model throughout the inference run. We compare that to inference with automated model switching per input enabled by *Sommelier*.

With *Sommelier* identifying tens of functionally equivalent models (pre-registered to the repository) with different resource profiles, the inference server can switch to more compact models when experiencing heavy loads and the highest quality models when the resource is abundant. Specifically, the automated model switching module of the inference serving engine first formulates a query (e.g., the one shown in Figure 6) combining the current and expected run-time conditions (e.g., memory availability, CPU shares, desirable accuracy), and the DNN model currently being served. Then, *Sommelier* executes this query to return a "similar" model that achieves the same expected accuracy as the original one but better matches the resource availability of the machine.

Model switching is orthogonal to and combinable with system optimizations like replication, parallelization, and batching. Therefore, we also emulate an ideal scenario of tail-latency reduction via system optimizations [16, 29, 63], by enlisting a standby inference server with identical resource setups. Under light loads, the inference tasks are submitted to both servers and the earlier completion is counted; Under heavy loads, the inference tasks are evenly distributed on both servers to fully utilize all available resource.

Figure 9(c) shows the inference latency distributions for the baseline, ideal system optimizations only, automated model switching with *Sommelier*, and combining *Sommelier* with system optimizations. Compared to the baseline, *Sommelier* could reduce the tail latency of inference tasks by a factor of 6 without using additional resources by switching models automatically, which is not possible for current model repositories and inference serving platforms. This compares to about 33% reduction with system optimizations alone. Moreover, *Sommelier* can work in tandem with existing system optimizations to further cut down tail latency by over 15%. Note that while the specific latency numbers and improvement ratios depend on the hardware used, the general trends hold, i.e., the benefit from model switching can be far more significant than common system optimizations. This is not surprising, since the execution time of DNN inference is inherently predictable [29]. Therefore, reducing the size of a DNN can predictably reduce the inference latency.

As more model variants are generated to match diverse resource consumption profiles orders of magnitude apart [30], there is significant untapped opportunity in leveraging model switching without scaling out. Models initially intended for resource-constrained edge devices can be useful in cluster settings at heavy loads[1].

Across the 200 models and all types of CV and NLP inference tasks, on average, there is only negligible difference in the inference accuracy due to automated model switching with *Sommelier*. We first calculate the replacement model's accuracy relative to the original model, and then consider the variation of that relative accuracy change. Across the inference task categories, the 90th percentile of the *relative* inference accuracy increase or drop is all between 1.7% and 2.4%.

7.2 Assessing functional equivalence

Between model segments. To evaluate how well *Sommelier* captures the equivalence between model segments used in transfer learning, we use the three CV tasks mentioned earlier, whose models are all transferred from the pre-trained ResNet50. We fine-tune each *original* model by freezing different numbers of base layers (mimicking different transfer attempts). Then, we replace the newly tuned model segment (i.e., layers) with the counterpart in the original one, and evaluate the new quality of result (QoR) relative to the pre-replacement original result quality (normalized to 100%). Separately, we add noise to the parameters of each *fine-tuned* model to mimic worst-case fine-tuning and generate the corresponding reference model. We then derive the theoretical quality lower bound from the reference and the segment-replaced models.

Figures 10(a) to 10(c) each plots three lines. The y-axis shows the QoR (i.e., accuracy) of the "partially replaced" model relative to the original model. The dashed line corresponds to the estimated lower-bound accuracy from our algorithm, whereas the two solid lines ("fine-tuned" and "noisy") capture the relative accuracy in normal and worst-case fine-tuning scenarios. The shaded region (accuracy loss within 10%) marks acceptable model replacement usage, and our algorithm generates reliable lower bounds that closely track the actual accuracy in this region.

Whole model equivalence. With respect to ResNet50 as the reference model, we calculate the bounds given different validation dataset sizes for Inception-V3, VGG19, and MobileNet, all three achieving the same image recognition functionality. The actual accuracy while interchanging these models for their tasks is empirically measured 20 times with the same validation dataset size and

[1]The overhead in GPU memory swap can be mitigated by switching models in the background [30].

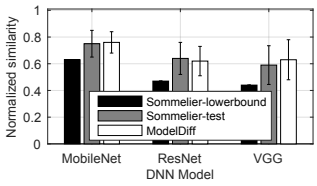
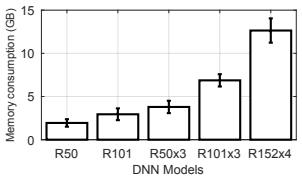
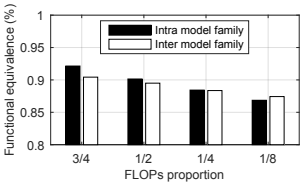


Figure 11: DNN similarity score comparison.



(a) Resource variation.



(b) Functional equivalence.

Figure 12: Effectiveness of resource and semantic indices.

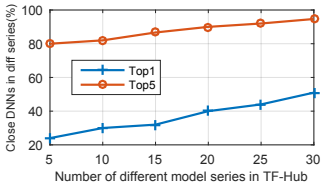


Figure 13: Cross-series DNN similarity.

Table 1: Lower bound vs actual accuracy (%). A cell (X/Y/Z) reports the "bound/min/average" of the actual accuracy.

Dataset Size	InceptionV3	VGG19	MobileNet
100	54 / 64 / 72	54 / 66 / 75	50 / 57 / 69
1k	62 / 68 / 72	62 / 70 / 75	58 / 66 / 69
10k	67 / 70 / 72	70 / 72 / 75	62 / 66 / 69

we compare the accuracy lower bound with the lowest and average actual accuracy values. Table 1 confirms the accuracy bound is safe and gradually approaches the actual accuracy when a larger validation dataset is used. When over 1000 records are used for validation, the bound is within 10% of the actual accuracy.

Comparison to ModelDiff. To the best of our knowledge, no existing work detects and exploits functional equivalence between *DNN segments* or builds a query engine based on whole model equivalence. The closest to *Sommelier* is a recent work, ModelDiff [48], which proposes a *metric* to quantify the similarity between *whole* DNN models, based on the cosine similarity between the decision distance vectors from two models; the metric is empirically obtained using a testing based approach trained on a benchmark set of DNN models and datasets, which is precisely what *Sommelier* tries to avoid. Thus, we compare the generalization bound approach in *Sommelier* and ModelDiff, to highlight the unique benefits of *extensional* measurements leveraging a generalization bound over an *intensional* testing-based approach. We follow the same settings as in the ModelDiff paper. We use three DNNs (i.e., MobileNet, ResNet, and VGG), fine-tune them to certain levels, select the testing dataset from ImageNet using the algorithm proposed in ModelDiff, and feed the same dataset to both *Sommelier* and ModelDiff to measure the functional equivalence (i.e., the level of similarity) between the original model and their fine-tuned variant. Each experiment is repeated 20 times with different datasets, and the error bars show the minimal and maximal values among these runs.

Figure 11 shows that both the "testing-only" *Sommelier* and ModelDiff can successfully detect the similarity between models, and there is no statistically significant difference in their **average** performance. However, *Sommelier* can further leverage the generalization bound analysis to lower-bound the model similarity scores. Without such a bound, the similarity scores given by ModelDiff can vary by around 30%, depending on the exact dataset used for the measurement process. This makes ModelDiff less useful than *Sommelier* for scenarios where predictability and safety are critical.

ModelDiff considers whole models only, even for a model pair transferred from the same base. In that case, how much these models deviate from each other would depend on the additional layers added to the common base, often trained specifically to "specialize" to the specific task. Functionally it is more appropriate to replace only the common base, not the additional layers. The problem with replacing the entire model instead will mainly manifest in certain inference results being widely wrong (i.e., the "corner cases"), and this is not always captured by the overall inference accuracy over some generic validation dataset.

7.3 Tensorflow Hub case study

We conduct a case study of 163 DNN models from 30 series in TensorFlow Hub [6]. Each series is a family of models derived from a common basis. Consider two widely used series, BiT [43] (5 models) and EfficientNet [70] (8 models). Each includes a sequence of increasingly large and accurate models, from tens to hundreds of millions of parameters, and from 74% to 86% in accuracy. Currently, manual model selection is done intra-series for simplicity.

First, we let *Sommelier* index the 13 models from BiT and EfficientNet only, shown in Figure 12. For the resource index, Figure 12(a) shows the variation in the memory consumption for the BiT models relative to the memory usage of a target setting (e.g., GPU specification and batch size). When the setting changes, the memory consumption of the model can vary by 25%, which would necessitate exhaustive, case-by-case profiling. In contrast, *Sommelier*'s resource index efficiently organizes models and accounts for diverse execution settings, obviating such a case-by-case search. For the semantic index, we use the largest BiT model (the R152x4 model) as the reference model, and assess the functional equivalence between the reference model and any model in the BiT and EfficientNet collections with a similar resource profile. Figure 12(b) surprisingly shows that, if we want a model that is one-eighth the size of R152x4 to replace R152x4, the better one is *from EfficientNet*, not from the same BiT collection. This is hard to identify manually.

Next, we incrementally index more series, eventually covering all 163 models. For each model, we identify its top-K functional equivalents (i.e., the models with the K highest semantic equivalence scores) within and across model series. In Figure 13, the x-axis shows how many randomly selected series are indexed, and the y-axis shows the portion of series with models finding the top-K semantic equivalents *outside* their own series (repeated 5 times). On average, up to 40% and 80% of the 30 series find the top-1 and top-5 functionally equivalent DNNs *in another series*. This confirms the widespread nature of hidden correlation between models, highlighting the value of automatic semantic assessment in *Sommelier*. Further considering partial model equivalence relations, the aforementioned percentage exceed 50% and 90% respectively even when only indexing up to 5 series. Regardless of the number of model series indexed, the inference result agreement ratios between the top-5 closest models are always well above each model's absolute inference accuracy, consistent with Figure 3.

Table 2: Time of functional equivalence check.

Metrics	AlexNet	ResNet	VGG19	BERT
# Params (M)	62	60	143	340
Time (Segment)	1.89s	2.77s	5.46s	14.10s
Time (Whole)	1.25s	4.46s	6.18s	22.92s

Table 3: Run-time query latency (ms).

Predicate	Number of records			
	100	1K	10K	100K
Resource	0.22	0.54	1.63	4.32
Semantic	0.01	0.03	0.04	0.06
Both	0.24	0.61	2.30	6.69

7.4 *Sommelier* system overhead

Sommelier introduces several query operations during run time. In this section, we profile them individually.

Latency of functional equivalence detection. Recall that *Sommelier* assesses functional equivalence between models offline (Section 4), which is not on the critical path of processing inference workloads. We mainly consider whether *Sommelier* can handle huge DNN models. We use four models (Table 2 column titles) as the inputs to test run the whole model and model segment equivalence detection algorithms respectively. Table 2 shows the sizes of the models and the time needed. We can clearly see that the algorithm scales well even when the model size is extremely large. For BERT (i.e., one of the largest commonly used DNN models), with over 340 million parameters and requiring over 12 GBs of memory during the run time, our algorithm still finishes within around 20 s, reasonable for offline index calculation.

Latency of run-time queries. The query operations are on the online path (Section 5). The main latency overhead is from searching the LSH-based resource index and the 2D pair-wise DNN semantic index. This is slower than the extremely fast (ns level), hash-based search for current, naive DNN model lookups. We prepare the model repository with different numbers of models varying from 100 to 100K. In each case, the storage is queried 20 times, and we time the average search latency when given *either* a resource or semantic constraint alone, as well as when given both constraints. Table 3 shows the average query time versus storage size. The query is fast enough even considering both search predicates. In practice, the repository size needed is mostly smaller than 100K model records, where around 6 ms is the typical retrieval latency, orders of magnitude faster than typical inference tasks.

Memory overhead. Since *Sommelier* leverages index structures to track the semantic correlation and resource profiles of DNN models (Section 5), additional memory consumption is therefore inevitable. However, this should be negligible since only the metadata of the models need to be kept in memory, whereas the models themselves still reside on disk. Table 4 shows the added memory consumption by randomly picking different numbers of DNN models and building the two index structures. The additional memory footprint is mostly under 80 MB, indeed negligible compared to the memory capacity of modern hardware. This also leaves space for caching the most frequently used models in memory to further mask the model loading latency from a (remote) disk.

Table 4: Memory footprint (MB) of the indices.

# Models	10	100	1k	10K	100K
Resource	0.001	0.008	0.091	0.87	6.5
Semantic	0.006	0.58	23	55	71

8 Related Work

Our work takes a leaf out of theoretical work on DNN semantic analysis and explanatory queries in database literature.

Semantic analysis of DNN models. Until recently, there was little consideration of harnessing the statistical nature of DNNs [44]. Relevant works studied the functional semantics of DNNs to ensure their robustness, safety, and interpretability. Reluplex [42], PRIMA [52], and others [9, 67] explore DNN formal semantics using SMT solvers and other verification techniques. Manifold [80], DeepTest [71] and DeepXplore [57] build systems to validate the robustness solely via iteratively refined test datasets. However, they all verify local properties (e.g., adding perturbation to an input) of DNNs, hence not applicable to task-level model semantics comparison. We discussed ModelDiff [48] in Section 7.2 for empirical whole model comparison. MLink [78], concurrent with our work, similarly tests holistic model correlation to optimize for inference serving. Recent work on interpretable AI (e.g., OMG [40] and ARG [23]) leverage continuity features of the input and human visual perception to *explain* why and how a given model matches the functional semantics of a DL task. Instead, *Sommelier* provides the "inverse" function, serving the optimal DNNs to users with DL tasks and performance goals at hand.

Explanation query engines. Recent database literature has explored database functionalities to detect causality, analyze inherent correlation within the data, and answer explanation queries for users. Relevant efforts include theoretical analysis frameworks [25, 65], relational interfaces for explanation [7], and optimizations for domain-specific data types (e.g., performance traces [76] and error logs [64]). Although these techniques are agnostic to the specific data type, they lack the capability to extract relevant information from DNN models to effectively handle explanation queries. *Sommelier* develops essential tools for this purpose.

9 Conclusion

DNN repositories are essential but currently require the user to profile and identify precisely which model to use. Instead, We propose *Sommelier*, an indexing and query system over typical DNN repositories, using a novel primitive to quantify functional equivalence between DNN models. *Sommelier* is built as a standalone query engine that can interface with an existing repository. Extensive evaluation shows that *Sommelier* can identify the ideal model for over 95% of the queries, and reduce the 90th percentile tail latency of inference tasks by a factor of 6 when interfaced with an inference server for automated run-time model switching. By abstracting away the model repository altogether, *Sommelier* can automate model management and fundamentally change the existing system pipelines for DNN inference, testing, and design.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and suggestions. This work is funded by the National Science Foundation under Grant Nos. CNS-1815115 and 2112562, and a Google Faculty Research Award.

References

[1] 2022. *Intel(R) Math Kernel Library for Deep Neural Networks (Intel(R) MKL-DNN).* https://intel.github.io/mkl-dnn/

[2] 2022. *ONNX: Open Neural Network Exchange Format.*

[3] 2022. *Open source deep learning code and pre-trained models.*

[4] 2022. *PyTorch Hub.*

[5] 2022. *Sommelier source code.*

[6] 2022. *TensorFlow Hub: A repository of reusable assets for machine learning with TF.*

[7] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, et al. 2018. DIFF: A relational interface for large-scale data explanation. *Proceedings of the VLDB Endowment* 12, 4 (2018), 419–432.

[8] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. 2018. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296* (2018).

[9] Mislav Balunovic and Martin Vechev. 2020. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations.*

[10] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. 2017. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems.* 6240–6249.

[11] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark analysis of representative deep neural network architectures. *IEEE Access* 6 (2018), 64270–64277.

[12] Han Cai, Chuang Gan, and Song Han. 2019. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).

[13] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. 2019. Scaling video analytics on constrained edge nodes. *SysML* (2019).

[14] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MxNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[15] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. 2017. Adanet: Adaptive structural learning of artificial neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70.* JMLR. org, 874–883.

[16] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 613–627.

[17] Bo Dai, Sanja Fidler, Raquel Urtasun, and Dahua Lin. 2017. Towards diverse and natural image descriptions via a conditional GAN. In *Proceedings of the IEEE International Conference on Computer Vision.* 2970–2979.

[18] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 11398–11407.

[19] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry.* ACM, 253–262.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 248–255.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[22] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. 2017. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 276–286.

[23] Upol Ehsan, Pradyumna Tambwekar, Larry Chan, Brent Harrison, and Mark O Riedl. 2019. Automated rationale generation: A technique for explainable AI and its effects on human perceptions. In *Proceedings of the 24th International Conference on Intelligent User Interfaces.* 263–274.

[24] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n.d.]. The Pascal Visual Object Classes (VOC) Challenge. ([n. d.]).

[25] Ronald Fagin, R Guha, Ravi Kumar, Jasmine Novak, D Sivakumar, and Andrew Tomkins. 2005. Multi-structural databases. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* 184–195.

[26] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (S&P).* IEEE, 3–18.

[27] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. 2018. MorphNet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1586–1595.

[28] Gregory Griffin, Alex Holub, and Pietro Perona. 2007. Caltech-256 object category dataset. (2007).

[29] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20).* 443–462.

[30] Peizhen Guo, Bo Hu, and Wenjun Hu. 2021. Mistify: Automating DNN Model Porting for On-Device Inference at the Edge. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21).*

[31] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. 2018. FoggyCache: Cross-Device Approximate Computation Reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking.* ACM, 19–34.

[32] Peizhen Guo and Wenjun Hu. 2018. Potluck: Cross-Application Approximate Deduplication for Computation-Intensive Mobile Applications. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, 271–284.

[33] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied machine learning at Facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 620–629.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision.* Springer, 630–645.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop.* http://arxiv.org/abs/1503.02531

[36] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[37] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. 2018. Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing. In *2018 USENIX Annual Technical Conference (ATC).* 29–42.

[38] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on.* IEEE, 1–12.

[39] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *PVLDB* 10, 11 (2017), 1586–1597.

[40] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. 2020. Model Assertions for Monitoring and Improving ML Model. *arXiv preprint arXiv:2003.01668* (2020).

[41] Hamid Karimi, Tyler Derr, and Jiliang Tang. 2019. Characterizing the Decision Boundary of Deep Neural Networks. *arXiv preprint arXiv:1912.11460* (2019).

[42] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification.* Springer, 97–117.

[43] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. 2019. Big Transfer (BiT): General Visual Representation Learning. (2019). arXiv:1912.11370 [cs.CV]

[44] Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, and Matei Zaharia. 2019. Willump: A Statistically-Aware End-to-end Optimizer for Machine Learning Inference. *arXiv preprint arXiv:1906.01974* (2019).

[45] Adarsh Kumar, Arjun Balasubramanian, Shivaram Venkataraman, and Aditya Akella. 2019. Accelerating deep learning inference via freezing. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19).*

[46] Yun Li, Chen Zhang, Shihao Han, Li Lyna Zhang, Baoqun Yin, Yunxin Liu, and Mengwei Xu. 2021. Boosting Mobile CNN Inference through Semantic Memory. In *Proceedings of the 29th ACM International Conference on Multimedia.* 2362–2371.

[47] Yun Li, Chen Zhang, Shihao Han, Li Lyna Zhang, Baoqun Yin, Yunxin Liu, and Mengwei Xu. 2021. Boosting Mobile CNN Inference through Semantic Memory. In *Proceedings of the 29th ACM International Conference on Multimedia (MM'21).*

[48] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. 2021. ModelDiff: Testing-based DNN similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 139–151.

[49] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision.* Springer, 740–755.

[50] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. 2018. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3238147.3238202

[51] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015

[52] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2022. PRIMA: general and precise neural network certification via scalable convex hull approximations. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–33.

[53] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. 2017. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564* (2017).

[54] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-serving: Flexible, high-performance ML serving. *arXiv preprint arXiv:1712.06139* (2017).

[55] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. *arXiv preprint arXiv:1811.09886* (2018).

[56] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[57] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[58] Hang Qi, Evan R Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks.. In *ICLR (Poster)*.

[59] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[60] Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. 2016. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 267–278.

[61] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. MLPerf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[62] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[63] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2019. INFaaS: A model-less inference serving system. *arXiv preprint arXiv:1905.13348* (2019).

[64] Sudip Roy, Arnd Christian König, Igor Dvorkin, and Manish Kumar. 2015. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 1167–1178.

[65] Sudeepa Roy and Dan Suciu. 2014. A formal approach to finding explanations for database queries. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1579–1590.

[66] Erik F Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050* (2003).

[67] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. 2019. Beyond the Single Neuron Convex Barrier for Neural Network Certification. In *Advances in Neural Information Processing Systems*. 15072–15083.

[68] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–30.

[69] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[70] Mingxing Tan and Quoc V Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[71] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[72] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. 2018. Transfer Learning with Neural AutoML. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 8366–8375. http://papers.nips.cc/paper/8056-transfer-learning-with-neural-automl.pdf

[73] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. 2010. Sun database: Large-scale scene recognition from abbey to zoo. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 3485–3492.

[74] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1492–1500.

[75] Neeraja J Yadwadkar, Francisco Romero, Qian Li, and Christos Kozyrakis. 2019. A case for managed and model-less inference serving. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 184–191.

[76] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. DBSherlock: A performance diagnostic tool for transactional databases. In *Proceedings of the 2016 International Conference on Management of Data*. 1599–1614.

[77] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[78] Mu Yuan, Lan Zhang, and Xiang-Yang Li. 2022. MLink: Linking Black-box Models for Collaborative Multi-model Inference. In *Proceedings of AAAI*.

[79] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems* 30, 9 (2019), 2805–2824.

[80] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. 2018. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics* 25, 1 (2018), 364–373.

[81] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. 2017. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2881–2890.

[82] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. 2017. Scene Parsing through ADE20K Dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.