Beimingwu: A Learnware Dock System

Zhi-Hao Tan[†] Nanjing University Nanjing, China
Jian-Dong Liu[†] Nanjing University Nanjing, China
Xiao-Dong Bi Nanjing University Nanjing, China
Peng Tan Nanjing University Nanjing, China
Qin-Cheng Zheng Nanjing University Nanjing, China
Hai-Tian Liu Nanjing University Nanjing, China
Yi Xie Nanjing University Nanjing, China
Xiao-Chuan Zou Nanjing University Nanjing, China
Yang Yu Nanjing University Nanjing, China
Zhi-Hua Zhou[*] Nanjing University Nanjing, China

ABSTRACT

The learnware paradigm proposed by Zhou [40] aims to enable users to leverage numerous existing high-performing models instead of building machine learning models from scratch. This paradigm envisions that: Any developer worldwide can submit their well-trained models spontaneously into a *learnware dock system* (formerly known as *learnware market*). The system uniformly generates a *specification* for each model to form a *learnware* and accommodates it. As the key component, a specification should represent the capabilities of the model while preserving developer's original data. Based on the specifications, the learnware dock system can identify and assemble existing learnwares for users to solve new machine learning tasks. Recently, based on reduced kernel mean embedding (RKME) specification, a series of studies have shown the effectiveness of the learnware paradigm theoretically and empirically. However, the realization of a learnware dock system is still missing and remains a big challenge.

This paper proposes `Beimingwu`, the first open-source learnware dock system, providing foundational support for future research. The system provides implementations and extensibility for the entire process of learnware paradigm, including the submitting, usability testing, organization, identification, deployment, and reuse of learnwares. Utilizing `Beimingwu`, the model development for new user tasks can be significantly streamlined, thanks to integrated architecture and engine design, specifying unified learnware structure and scalable APIs, and the integration of various algorithms for learnware identification and reuse. Notably, this is possible even for users with limited data and minimal expertise in machine learning, without compromising the raw data's security. The system facilitates the future research implementations in learnware-related algorithms and systems, and lays the ground for hosting a vast array of learnwares and establishing a learnware ecosystem. The system is fully open-source and we expect the research community to benefit from the system. The system and research toolkit have been released on GitLink [1] and GitHub [2].

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Information systems** → **Information systems applications**.

KEYWORDS

Machine Learning, Learnware, Learnware Dock System, Learnware Specification

ACM Reference Format:
Zhi-Hao Tan, Jian-Dong Liu, Xiao-Dong Bi, Peng Tan, Qin-Cheng Zheng, Hai-Tian Liu, Yi Xie, Xiao-Chuan Zou, Yang Yu, and Zhi-Hua Zhou. 2024. Beimingwu: A Learnware Dock System. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3637528.3671617

1 INTRODUCTION

Numerous applications that rely on machine learning models have been integrated into many facets of modern life. However, in classic machine learning paradigm, to train a high-performing model from scratch for a new task still requires an abundance of high-quality data, expert experience, and computational resources, which is difficult and expensive. There are also lots of concerns when reusing existing efforts, such as the difficulty of adapting a specific trained model to different environments, and the embarrassment of catastrophic forgetting when refining a trained model incrementally. Besides, privacy and proprietary issues hinder the data sharing among developers, and restrict the capabilities of big models in many data-sensitive scenarios. Indeed, most efforts have been focusing on one of these concerned issues separately, paying less attention to the fact that most issues are entangled in practice.

To tackle the above issues simultaneously and leverage existing efforts in a systematic and unified way, *learnware* [40, 41] was proposed, based on which machine learning tasks can be solved in a novel paradigm. The core design is envisioned as follows: For well-trained models of any structure from various tasks, a learnware consists of the model itself and a *specification* which captures the model's specialty in a certain representation, like its statistical properties. Developers worldwide can submit their trained models into a *learnware dock system* spontaneously, and the system helps generate specifications for each model to form learnwares.

[*]Corresponding author (email: zhouzh@lamda.nju.edu.cn). All authors are affiliated with National Key Laboratory for Novel Software Technology and School of Artificial Intelligence in Nanjing University.

This work was supported by NSFC (62250069).

[†]Equal contribution

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671617

[1]https://www.gitlink.org.cn/beimingwu
[2]https://github.com/Learnware-LAMDA

By accommodating all learnwares, when facing a new user task, the learnware dock system can identify and assemble useful learnware(s) based on specifications. Instead of starting from scratch, the user can apply these learnware(s) directly or adapt by her own data for better usage. Note that the learnware dock system should be able to preserve the raw data of model developers and users.

As the foundation of learnware paradigm, a learnware dock system should uniformly accommodate numerous submitted models, and leverage the capabilities of them to solve new tasks in a unified way. The key problem is that, considering a learnware dock system which has accommodated thousands even millions of models, how to identify and select the most helpful learnware(s) for a new user task? Apparently, direct submitting user data to the system for trials would be unaffordable and leak user's raw data. The core design of learnware paradigm lies in the specification. Recently, the reduced kernel mean embedding (RKME) specification [41] was proposed, which captures the distribution information via a synthetic reduced set. Based on the RKME specification, multiple learnware search and reuse algorithms are proposed and the effectiveness of specification-based model selection and combination is verified empirically and theoretically [14, 25, 26, 30, 32, 35]. However, the realization of an initial learnware dock system is still missing and remains a big challenge, which needs a novel specification-based architecture design to support various research algorithm implementation.

To establish the foundation for the future research of learnware paradigm, we built `Beimingwu`, the first open-source learnware dock system. Benefiting from scalable system and engine architecture design, specifying unified learnware structure and scalable APIs, integration of baseline algorithms for the entire process, and construction of convenient algorithm evaluation scenarios, the system not only facilitates future research in learnware-related algorithms, but also lays the groundwork for hosting a vast array of learnwares and establishing a learnware ecosystem. In this paper, our contributions can be summarized as follows:

- Based on the first systematic implementation of learnware dock system, as envisioned in the learnware paradigm, `Beimingwu` can significantly streamline the process of building machine learning models for new tasks, even with limited data and minimal expert knowledge, while ensuring data privacy.
- In `Beimingwu`, we specify a unified learnware structure, and design an integrated architecture for system engine, which can support the entire process including the submitting, usability testing, organization, identification, deployment, and reuse of learnwares. The architecture is scalable to various models, and possesses unified and scalable interfaces for future research in learnware-related algorithms.
- Based on the engine architecture and RKME specification, we implement and refine a set of baseline algorithms for specification generation, learnware organization, identification, and reuse. The engine is released as `learnware` package, which supports the computational and algorithmic aspects of the system, and also serves as a research platform of learnware paradigm.
- To realize a learnware dock system that operates stably online, based on the engine, we further design, develop, and deploy the system backend and user interface including web frontend and command-line client. All the source code of the system frontend, backend, and engine is open-source for collaborative community contributions and convenience of learnware research.
- Benefiting from the RKME statistical specification, we introduce a baseline method for organizing, identifying, and reusing learnwares from different feature spaces.
- We build various types of experimental scenarios and conduct corresponding empirical studies for evaluation, which are all public for future research.

2 A BRIEF REVIEW OF LEARNWARE PARADIGM

The learnware paradigm was proposed in [40], and many progresses have been summarized in [41]. "Learnware = Model + Specification", where the model is any well-trained machine learning model, and the specification offers a certain kind of characterization for the model and enables the model to be adequately identified to reuse according to future users' requirement.

The developer or owner of a well-performing machine learning model, of any type and structure, can spontaneously submit her trained model into a *learnware dock system* (previously called *learnware market*). If the model passes the quality detection, the learnware dock system will help generate a specification to the model and accommodate it in the system in a unified way. The learnware dock system should be scalable to accommodate thousands of or millions of well-performed models submitted by different developers, on different tasks, using different data, optimizing different objectives, etc.

Based on the learnware dock system, when a user is going to solve a new machine learning task, she can submit her *requirement* to the learnware dock system, and then the system will identify and assemble some helpful learnware(s) from numerous learnwares to return to the user by considering the learnware specification. She can apply the learnware(s) directly, or adapt them by her own data, or exploit in other ways to help improve the model built from her own data. No matter which learnware reuse mechanism is adopted, the whole process can be much less expensive and more efficient than building a model from scratch by herself. Importantly, it has been proved that the learnware paradigm has privacy-preserving ability, which enables developers to share their models that can be adequately identified and reused by future users without disclosing developer's original training data.

The learnware paradigm proposes to build a learnware dock system to accommodate, organize and leverage existing well-performing models uniformly, which provides a unified way to leverage existing efforts from all the community to solve new user tasks, and offers the possibility of addressing significant concerned issues simultaneously [41]: lack of training data, lack of training skills, catastrophic forgetting, hard to achieve continual learning, data privacy/proprietary, unplanned new tasks from open world, and carbon emissions caused by wasteful repetitive training.

The key challenge is that, considering a learnware dock system which has accommodated millions of models, how to identify and assemble the most helpful learnware(s) for a new user task? Apparently, direct submitting user data to the system for trials would be unaffordable and leak user's raw data. The core design of learnware paradigm lies in the specification. Recent progresses are mainly

based on the RKME specification [41]. For example, Wu et al. [30] proposed to identify helpful learnware(s) by matching original data distributions of learnwares with user data distributions based on RKMEs, which is further extended by Zhang et al. [35] under the existence of unseen parts in user tasks. By learning a unified specification space for various learnwares from heterogeneous feature spaces, Tan et al. [25, 26] proposed learnware search and reuse algorithms to utilize the learnware from heterogeneous feature spaces. To support efficient and accurate identification from a large number of learnwares, Xie et al. [32] proposed the anchor-based mechanism, which organizes learnwares structurally and identifies helpful learnwares by only accessing a small number of anchor learnwares instead of examining all learnwares. Liu et al. [14] further proposed an efficient learnware identification method called evolvable learnware specification with index (ELSI), resulting in increasingly accurate characterization of model abilities beyond original training tasks with the ever-increasing number of learnwares. Besides, Guo et al. [6] attempted to leverage learnwares from heterogeneous label spaces.

Although existing study has shown the effectiveness of the learnware paradigm, the realization of a learnware dock system is still missing and remains a big challenge, which needs a novel specification-based architecture design to handle the diversity of real-world tasks and models, and to leverage numerous learnwares in a unified way according to the user task requirement. In this paper, we built `Beimingwu`, the first learnware dock system, which can support the entire process including the submitting, usability testing, organization, management, identification, deployment, and reuse of learnwares. Based on `Beimingwu`, we lay the groundwork for future research and ecosystem, and show the effectiveness of learnware paradigm in addressing concerned issues in Section 1 simultaneously.

3 SOLVING LEARNING TASKS VIA BEIMINGWU

Benefiting from uniform learnware structure, integrated architecture design, and unified interfaces, all learnwares in `Beimingwu` can be uniformly identified and reused. Based on the first systematic implementation of learnware dock system, as envisioned in the learnware paradigm, `Beimingwu` can significantly streamline the process of building machine learning models for new tasks: Excitingly, given a new user task, if `Beimingwu` possesses learnwares with capabilities to tackle the task, with just a few lines of code, the user can easily obtain and deploy a high-quality model based on `Beimingwu`, without the need for extensive data, expert knowledge, or revealing her raw data.

Users can also interact with the system via a web frontend [3], to view information of each learnware and obtain helpful learnwares by choosing semantic specification, like data type, task type, and scenarios, or by uploading statistical RKME specification for precise identification. Note that the core engine of `Beimingwu`, supporting the computational and algorithmic aspects, is extracted and released as `learnware` package that can be easily used locally.

The entire workflow of using `Beimingwu` is shown in Figure 1. Based on engineering implementations and unified interface design, each step can be achieved with one key line of code. Note that the

[3]https://bmwu.cloud/

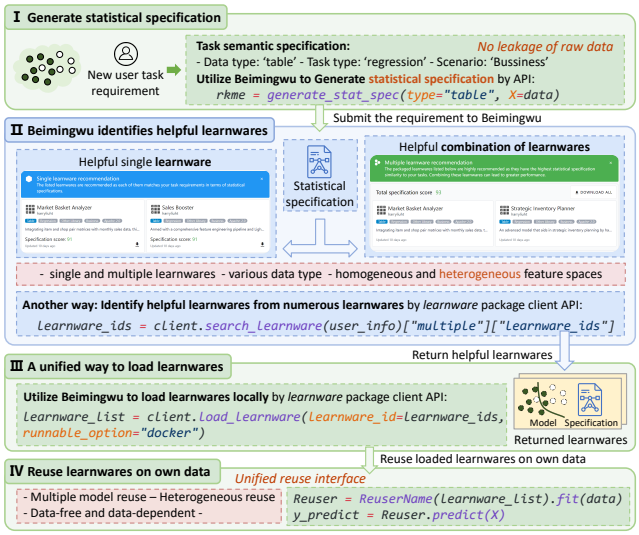

Figure 1: The entire workflow of using `Beimingwu` to solve new learning tasks includes statistical specification generation and learnware identification, loading, and reuse.

statistical specification is realized via reduced kernel mean embedding (RKME) specification [41] which captures the data distribution while not disclosing the raw data. Detailed learnware identification algorithms based on statistical specifications and learnware reuse algorithms will be presented in Section 5.1.

Based on `Beimingwu` system, we show that the learnware paradigm offers a promising solution to the issues mentioned in Section 1. Specifically, the model development process for tasks based on `Beimingwu` possesses the following significant advantages:

- **No need for extensive data and training resources.** If helpful learnwares exist, `Beimingwu` identifies and assembles helpful learnware(s) for user tasks from numerous learnwares in the system, then users can directly utilize them or refine them with a small amount of data, instead of training a new model with extensive data and resources from scratch.
- **Minimal machine learning expertise.** With just a few lines of code, users can easily obtain suitable learnware(s) identified by the system for their specific tasks. This streamlined process makes numerous high-quality and potentially helpful models adequately utilized by users across all levels of expertise. It eliminates the need for expert knowledge in designing priors, or manually selecting algorithms and models.
- **Simple and secure local deployment of diverse models.** Ideally, the system accommodates diverse and high-quality learnwares from global developers, applicable to various specialized and customized scenarios. Based on engineering implementations and architecture optimizations, and specifying a unified learnware structure, `Beimingwu` allows for effortless and safe deployment and reuse of arbitrary learnwares in a unified way based on containerized isolation, with fewer concerns about environment compatibility and safety.
- **Privacy-preserving: no leakage of original data.** To identify the most suitable learnwares, a user generates and submits the

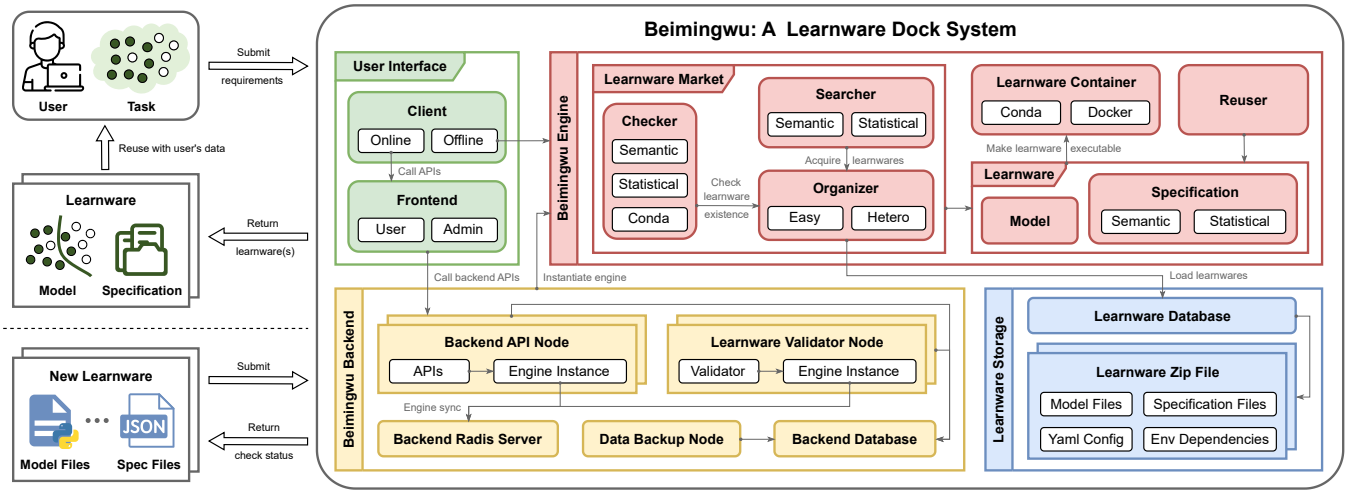

Figure 2: The architecture of Beimingwu consists of four layers: learnware storage, system engine, backend, and user interface.

RKME statistical specification to the system using API, which captures the data distribution while not disclosing the raw data. Based on the RKME, Beimingwu identifies the learnware(s) that are most beneficial for user task. More importantly, the privacy-preserving ability enables developers to share their models that can be adequately identified and reused by future users without disclosing developers' original training data.

Presently, at the initial stage, Beimingwu houses only about 1200 learnwares mostly built from open-source datasets, offering limited capabilities for numerous specific and unforeseen scenarios. However, relying on the foundational implementations and scalable architecture, the constantly submitted learnwares and algorithmic advancement will expand the knowledge base of the system and enhance its ability to reuse existing models to solve user tasks even beyond their original purposes, and this continuous evolution of the system enables it to handle increasing user tasks without catastrophic forgetting, naturally realizing lifelong learning.

4 DESIGN AND ARCHITECTURE

In this section, we will first provide an overview of the Beimingwu architecture. As briefly depicted in Figure 2, the entire system comprises four layers: learnware storage, system engine, system backend, and user interface. Then we will present our specification-based architecture design of the core engine of the system.

4.1 Overview of Beimingwu Architecture

Learnware storage layer. In Beimingwu, learnwares are stored as zip packages based on the architecture outlined in Section 4.2, containing model, specification, environment dependencies, and configuration files. To manage these zip packages, the learnware database stores crucial learnware-related information and provides a unified interface for the core engine of Beimingwu to access. Besides, the database can be constructed using either SQLite (suitable for easy setup in development and experimental environments) or PostgreSQL (recommended for stable deployment in production environments), both utilizing the same management interface.

Core engine layer. To maintain the simplicity and structure of Beimingwu, we have separated the core components and algorithms from the extensive engineering details. These extracted elements are now available as `learnware` package, which serves as the core engine of Beimingwu. As the system kernel, the engine encompasses all processes within learnware paradigm, including the submitting, usability testing, organization, identification, deployment, and reuse of learnwares. It operates independently of the backend and frontend, offering rich algorithmic interfaces for learnware-related tasks and research experiments. Moreover, *specification* serves as the central component in the engine, characterizing the associated model from both semantic and statistical perspectives, and connecting various essential learnware-related components. More details about the core engine are presented in Section 4.2.

In contrast to existing model management platforms, which passively collect and store models, Beimingwu actively manages learnwares through its engine. The system organizes learnwares via specifications, identifies relevant learnwares for users, and provides corresponding methods for learnware reuse and deployment.

System backend layer. To enable industrial-level deployment of Beimingwu, we have developed the system backend, building upon the core engine. Through the design of multiple modules and extensive engineering development, Beimingwu is now capable of online and stable deployment, providing comprehensive backend APIs to both the frontend and clients. Besides, to ensure efficient and stable system operation, we have implemented several engineering optimizations in the system backend layer, for example:

- **Asynchronous learnware validation.** Synchronous validation responses to concurrent learnware uploads can severely impact system efficiency. To address this challenge and efficiently validate submitted learnwares, we have designed an asynchronous validator node, which processes learnwares with a "waiting" status in a queue, significantly reducing system stress.
- **High concurrency across multiple backend nodes**: The API and validator nodes in the backend run concurrently, with their engine instances synchronized via a Redis server. When one

instance completes a learnware-related write operation, it notifies the others to reload the specific learnware using Redis, ensuring synchronization across all nodes.
- **Interface-level permission management**: For effective system management and convenient usage, we have designed three levels of permissions for backend APIs: no login needed, login required, and administrator permissions.
- **Backend database read-write separation**: `Beimingwu` employs a master-slave database architecture, separating read and write operations to different database instances, enhancing database and system efficiency.

User interface layer. For user convenience, we have developed a user interface layer, including a web-based frontend and a command-line client. The web-based frontend serves both user and administrative requirements and supports multi-node deployment, while the command-line client is seamlessly integrated into the `learnware` package, enabling users to access online APIs as well as learnware-related modules and algorithms.

4.2 Engine Architecture Design

In this section, we introduce the `Beimingwu` engine's architecture design. We will start with design principles and then delve into the engine architecture, covering modules and processes.

4.2.1 Design Principles. We design the `Beimingwu` engine based on the following guidelines that include decoupling, autonomy, reusability, and scalability.

Decoupling. As shown in Section 4.1, the `Beimingwu` engine is decoupled from the backend, focusing solely on learnware-related algorithms and components, while the backend is responsible for business logic and engineering optimization. This separation streamlines the system, enhancing clarity and maintainability by isolating academic modules from business and engineering components.

Autonomy. As an autonomous entity, the engine serves both as the system kernel and as a standalone research platform, covering all learnware-related processes and offering interactive interfaces. This allows for direct testing of new algorithms, avoiding the system's complex engineering and deployment hurdles.

Reusability. The engine incorporates various learnware components, covering submission, organization, identification, deployment, and reuse, designed with modularity to ensure each component handles specific tasks while promoting code reusability. This architecture principle allows for flexible module combinations, streamlining the creation of diverse learnware processes.

Scalability. As learnware research advances, the engine will integrate more algorithms, necessitating a highly scalable architecture. Modules should provide extension interfaces to accommodate future advanced algorithms for organization, identification, and reuse. This will also enable users to seamlessly incorporate and test their novel algorithms on this academic experimental platform.

4.2.2 Design of Core Modules. Following the design principles, we have created learnware, market, specification, model, reuse, and interface modules for the engine, as depicted in Figure 3.

Learnware. The learnware module, consisting of an identifier, specification, and user model modules, is created by parsing files that follow the learnware standard, including configuration, model, statistical and semantic specification, and environment files, as depicted in Figure 3. Specifically, the configuration file assists in module parsing and instantiation; the model file details the user model, offering a unified interface for training, prediction, and fine-tuning; the statistical and semantic specification files store statistics (e.g., training data distribution) and semantic information (e.g., task, data and scenario types) related to the model; the environment file specifies the user model's runtime dependencies.

Market. The market module, featuring an organizer, searcher, and multiple checkers, enables learnware organization, identification, and usability testing. (a) The organizer module focuses on learnware organization, facilitating operations like learnware reloading, insertions, deletions, updates, and evolving. (b) The searcher module identifies learnware(s) using statistical and semantic specifications, filtering and matching based on the similarity between multifaceted user requirements and learnware specifications. (c) The checker module assesses learnware usability and quality, verifying specifications and creating a runtime environment for testing user models.

Specification. The specification module, characterizing models from both semantic and statistical perspectives, serves as the engine's core component, facilitating learnware organization, identification, and reuse. The key lies in the statistical submodule, comprising user and system statistical specifications. (a) User specifications are generated locally by users via the engine, which currently supports reduced kernel mean embedding (RKME) specification [41] generation for tabular, image, and text data. The same type of RKME specifications can be compared for similarity using maximum mean discrepancy (MMD) calculations. During learnware search, the system identifies learnware(s) based on the similarity of statistical specifications. (b) System specifications are automatically generated by the system. For newly inserted learnwares, the organizer leverages existing user specifications to generate system specifications, enhancing learnware management and further characterizing their capabilities. Currently, the system employs heterogeneous mapping specifications in its system statistical module. When inserting learnwares from heterogeneous tabular data, the organizer module maps the user-generated RKME specifications to the same embedding space with heterogeneous mapping to support search operations on heterogeneous tabular models. For specific details about heterogeneous search, please refer to Section 5.2.

Model. The model module comprises the base model template and the model container. (a) The base model template standardizes training, prediction, and fine-tuning interfaces for user models. User models inherit from this template, enabling a unified interface for model operations. (b) The model container, also inherited from the base model template, automatically establishes an isolated runtime environment based on the environment file to execute models.

Reuse. The reuse module comprises the data-free reuser, data-dependent reuser, and aligner. Currently, the data-free reuser derives the final prediction directly, employing methods such as ensemble multiple learnwares. The data-dependent reuser, utilizing additional labeled data provided by users, obtains final prediction results through methods such as fine-tuning a meta-model. Both methods require input learnwares to have the same feature and prediction dimensions. The aligner ensures that features and prediction spaces of heterogeneous learnwares are mapped to a common

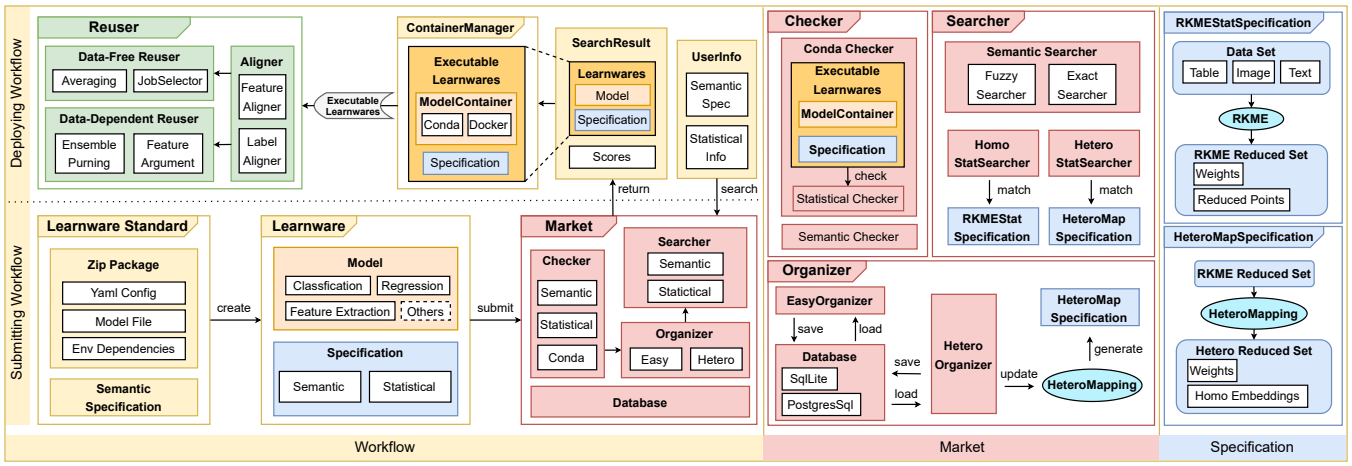

Figure 3: Architecture design of `Beimingwu` engine, encompassing both workflows and modules.

space. To handle learnwares with different dimensions, by leveraging the RKME specifications of obtained learnwares, the dock system passes them through the aligner to obtain feature-aligned learnwares, which can then be used by either the data-free reuser or data-dependent reuser. For specific details regarding the implementation of the aligner, please refer to Section 5.

4.2.3 Submitting and Deploying Stages. To provide a better understanding of the engine architecture's data flow and operations, we will focus on the processes in the submitting and deployment stages, as illustrated in Figure 3.

Submitting stage. In the submitting stage, learnware developers submit their learnware zip packages in the standard format to the system. The system then parses these packages into learnware instances, checks their usability and quality via the checker module, and stores the approved learnwares by the organizer module.

Deploying stage. In the deployment stage, users locally generate statistical and semantic specifications for their tasks and submit them to the system. The system then identifies learnware(s) that match the user requirements. Finally, based on the provided `learnware` package, users deploy these identified learnware(s) with isolated containers, and reuse them with various methods.

5 ALGORITHMS AND METHODOLOGIES

In this section, based on the integrated and scalable engine architecture and RKME specification, we then implement and refine a set of baseline algorithms for learnware specification generation, organization, identification, and reuse. Additionally, we propose a baseline method for the organization, identification, and reuse of learnwares from different feature spaces. The engine is released as `learnware` package[4], supporting the computational and algorithmic aspects of the system, and facilitates research implementation.

5.1 Implemented Algorithms of `Beimingwu`

Specifications. Specifications implemented by the system are all derived from the RKME specification [41], which uses techniques based on the reduced set of kernel mean embedding (KME) [2, 23]. Suppose a developer submits a model trained from the dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, then the reduced set representation $\{(\beta_j, \mathbf{z}_j)\}_{j=1}^n$ is generated by minimizing the distance between the KMEs of the reduced set and the original data measured by the RKHS norm: $\min_{\boldsymbol{\beta}\geq\mathbf{0},\mathbf{Z}} \left\| \frac{1}{m}\sum_{i=1}^m k(\mathbf{x}_i,\cdot) - \sum_{j=1}^n \beta_j k(\mathbf{z}_j,\cdot) \right\|_{\mathcal{H}}^2$. The RKME specification $\Phi(\cdot) = \sum_{j=1}^n \beta_j k(\mathbf{z}_j,\cdot) \in \mathcal{H}$ offers a concise representation of the original data distribution $\mathcal{P}$ while preserving data privacy.

The `Beimingwu` system provides a unified interface `generate_stat_spec` for model developers and users to easily generate specifications for specific data types. This generation process runs locally and does not involve any data sharing with the online system, thus ensuring data privacy and ownership. The system supports specifications for various data types:

- **Tabular specification:** For tabular tasks, the system generates two distinct types of specifications. The first, `RKMETableSpecification`, implements the RKME specification, which is the basis of tabular learnwares. It facilitates learnware recommendation and reuse for homogeneous tasks with identical input and output domains. The second, `HeteroMapTableSpecification`, enables learnware to support tasks with various input and output domains. This specification is derived from `RKMETableSpecification` and is produced using the system's heterogeneous engine. This engine, a tabular network, is trained on feature semantics and statistics of all tabular learnwares in the system.
- **Image specification:** The specification for image data `RKMEImageSpecification` introduces a new kernel function that transforms images implicitly before RKME calculation. It employs the neural tangent kernel (NTK) [7, 18, 27, 28], a theoretical tool that characterizes the training dynamics of deep neural networks in the infinite width limit, to enhance the measurement of image similarity in high-dimensional spaces.
- **Text specification:** Text inputs of varying lengths are processed into sentence embeddings using multilingual embedding models. Subsequently, the RKME specification `RKMETextSpecification` is calculated based on these embeddings.

[4] https://learnware.readthedocs.io/

Identification algorithms. When a user submits her task requirement, the system identifies useful learnware(s) by leveraging learnware specifications and the user's requirement. The task requirement typically includes semantic aspects based on tags and descriptions and statistical requirements based on RKME. Initially, the system executes a semantic filter across all learnwares, followed by a basic statistical search, which supports both single and multiple learnware searches.

- **Single learnware identification**: The system recommends learnware with data distribution similar to the user task. In details, given the learnware RKME specification $\mathbf{s}_l = \{(\beta_{li}, \mathbf{z}_{li})\}_{i=1}^{n_l}$ and user's RKME requirement $\mathbf{s}_u = \{(\beta_{ui}, \mathbf{z}_{ui})\}_{i=1}^{n_u}$, the system calculates the maximum mean discrepancy (MMD) distance between them: $\left\|\sum_{i=1}^{n_l} \beta_{li} k\left(\mathbf{z}_{li}, \cdot\right) - \sum_{j=1}^{n_u} \beta_{uj} k\left(\mathbf{z}_{uj}, \cdot\right)\right\|_{\mathcal{H}}^2$. The system transforms distances to scores and recommends learnwares with the score surpassing a preset threshold to the user.
- **Multiple learnware identification**: When a single learnware inadequately addresses the user's task, the system attempts to assemble a combination of learnwares for user's task. This involves using a weighted combination of data distributions from the learnwares to approximate the user's task data distribution. The system calculates the mixture weights for filtered learnwares and return learnwares with high weights.

Reuse algorithms. Upon identifying and providing users with pertinent learnwares, the system offers various basic methods for learnware reuse. These methods enable users to effectively apply learnwares to their tasks, thereby eliminating the need to develop models from scratch. There are two main categories:

- **Data-free reusers: reuse learnwares directly**. The average ensemble [39] used by `AveragingReuser` uniformly averages learnwares prediction, and the job selector [30] used by the `JobSelectorReuser` trains a multi-class classifier to identify the appropriate learnware for each user data.
- **Data-dependent reusers: reuse learnwares with minor labeled data**: The ensemble pruning [31] used by the `EnsemblePruningReuser` selects a subset from a given learnware list using multi-objective evolutionary algorithm and uses average ensemble. The feature augmentation used by the `FeatureAugmentReuser` enhances user task features by incorporating predictions from learnwares and subsequently training a simple model.

5.2 Handling Tabular Learnwares with Heterogeneous Feature Spaces

The specification, representing the model's specialty and utility, is a crucial component of the learnware. Each *specification island* encompasses all models that share the same functional space $\mathcal{F}: \mathcal{X} \mapsto \mathcal{Y}$ w.r.t. obj., where $\mathcal{X}$ represents the input domain, $\mathcal{Y}$ the output domain, and obj the objective. Collectively, these specification islands constitute the *specification world* [41]. When a user submits her task requirement, the system finds a matched specification island based on the user's input and output domains, followed by a more precise model identification based on specifications. However, for tabular tasks, a matched specification island may often not exist due to highly structured yet flexible data and complicated feature semantics, resulting in models from heterogeneous feature spaces.

To address these tabular task challenges, the system needs to merge different specification islands without having additional co-occurrence data across feature spaces. Inspired by Tan et al. [25] and Wang and Sun [29], our solution involves creating relationships between these islands based on RKME specifications and feature semantics through semantic embeddings. A heterogeneous tabular network is employed to establish this unified semantic embedding space, generating new specifications for each tabular learnware. These new specifications retain coefficients and transformed samples. This approach allows identifying learnwares from heterogeneous feature spaces.

Recommend heterogeneous tabular learnwares. To recommend potentially useful tabular learnware(s) from the entire collection, the system must integrate different tabular specification islands by establishing relationships between them. Notably, each specification island corresponds to a unique feature space $\mathcal{X}$, characterized by specific feature descriptions. The key strategy for merging these islands involves leveraging their relationships through feature semantics to create a unified specification world, based on semantic embeddings. In particular, the system employs a heterogeneous tabular network [29] to develop this unified semantic embedding space. This network serves as the system's heterogeneous engine, generating new specifications for each tabular learnware. For a given tabular learnware with specifications $\{(\beta_i, \mathbf{z}_i)\}_{i=1}^{n}$, the new specification retains the coefficient β_i and transforms the samples $\mathbf{z}_i$ using the system engine $F(\cdot)$, resulting in the new specification $\{(\beta_i, F(\mathbf{z}_i))\}_{i=1}^{n}$. Once all tabular specification islands are merged into this unified world, all specifications reside in the same space, enabling the tabular learnware recommendation to encompass all tabular learnwares, rather than just those with matching input and output domains.

Reuse heterogeneous tabular learnwares. Upon receiving the heterogeneous learnware recommended by the system, users are still unable to apply it directly to their tasks due to discrepancies in input domain $\mathcal{X}$ and output domain $\mathcal{Y}$. Nevertheless, the system facilitates the reuse of heterogeneous learnware through a three-step process: 1) aligning the input domain, 2) predicting with the learnware, and 3) aligning the output domain.

In the first step, input alignment, only the RKME specification of the recommended learnware and the RKME requirement are utilized, ensuring the user's original data remains confidential. This step involves transforming the feature space based on a reduced set of learnware specifications, denoted as $\mathbf{s}_l = \{(\beta_{li}, \mathbf{z}_{li})\}_{i=1}^{n_l}$, and a corresponding set of user requirements, $\mathbf{s}_u = \{(\beta_{ui}, \mathbf{z}_{ui})\}_{i=1}^{n_u}$. The transformation, represented by ϕ, minimizes the distance between two distributions: the learnware specification $\mathbf{s}_l = \{(\beta_{li}, \mathbf{z}_{li})\}_{i=1}^{n_l}$ and the projected user requirement $\mathbf{s}_u^{\text{proj}} = \{(\beta_{ui}, \phi(\mathbf{z}_{ui}))\}_{i=1}^{n_u}$. Utilizing the maximum mean discrepancy (MMD) distance, ϕ is derived from the following optimization problem, which is solved using gradient descent: $\min_{\phi} \left\|\sum_{i=1}^{n_u} \beta_{ui} k\left(\phi(\mathbf{z}_{ui}), \cdot\right) - \sum_{j=1}^{n_l} \beta_{lj} k(\mathbf{z}_{lj}, \cdot)\right\|_{\mathcal{H}}^2$.

With ϕ generated, the user's data is transformed to align with the learnware's input domain, enabling prediction. However, since the output domain of the user's task might differ from that of the learnware, output alignment is required. This process involves using a small amount of labeled data and is conducted by augmenting features, i.e., incorporating learnware predictions as additional

features to train a simple model, such as logistic regression for classification tasks and ridge regression for regression tasks.

6 EXPERIMENTAL EVALUATION

In this section, we first build a benchmark experimental scenario of tabular data, which contains hundreds of sales forecasting learnwares from various real-world stores in the system, and hundreds of new stores data as user tasks for evaluation, from heterogeneous feature spaces. Then we present empirical evaluations of the implemented baseline algorithms in Section 5 across various experimental scenarios. Finally, since our system has been open online, we will introduce the system operation situation and applications.

For simplicity, all experimental results (Figures 4–7) follow a consistent display format: The left table shows the results without labeled data, while the right figure depicts outcomes with varying amounts of labeled data, all derived from multiple experiments with the mean and standard deviation of average losses across all users.

6.1 Experiments on Tabular Data

Settings. The experimental scenario is built from three representative sales forecasting datasets: Predict Future Sales (PFS) [10], M5 Forecasting (M5) [16], and Corporacion [9], all experiments are public. Various feature engineering methods are used. LightGBM models are trained on the Corporacion and PFS training sets, with test sets and M5 datasets used as user tasks. The experimental scenario, containing 265 learnwares across five feature spaces and two label spaces, are all uploaded to `Beimingwu`.

Method	Loss
Mean in System	0.897
Best in System	0.756
Top-1	0.830
JobSelector	0.848
AverageEnsemble	0.816

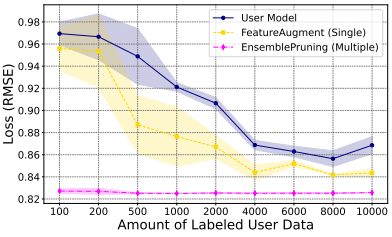

Figure 4: Results of homogeneous tabular experiments.

6.1.1 Homogeneous Cases. In the homogeneous cases, the 53 stores in the PFS dataset act as individual users, using their test data as user data and applying the same feature engineering approach used in the system. These users then search for homogeneous learnwares in the system with matching feature spaces. We compare various baseline algorithms under conditions of no or limited labeled data. Top-1 reuser employs the best single learnware identified by the searcher, while other reuse methods are detailed in Section 5.1. Fig. 4 displays the results: the left table shows that data-free reusers outperform random selecting a learnware from the system, while the right figure indicates that identifying and reusing single or multiple learnwares yields better results than self-trained models with limited training data.

6.1.2 Heterogeneous Cases. Based on the similarity between tasks handled by learnwares in the system and user tasks, heterogeneous cases can be further categorized into two types.

Method	Loss
Mean in System	1.149
Best in System	1.038
Top-1	1.075
AverageEnsemble	1.064

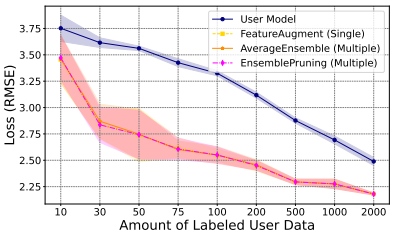

Figure 5: Results of heterogeneous tabular experiments.

Method	Loss
Mean in System	0.655
Best in System	0.304
Top-1	0.406
JobSelector	0.406
AverageEnsemble	0.310

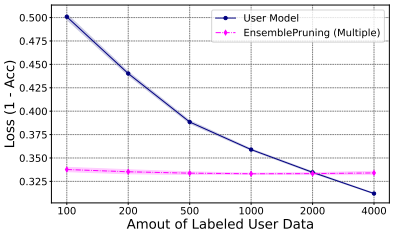

Figure 6: Results of image experiments.

Different feature engineering scenarios. We treat the 41 stores within the PFS dataset as users, creating the user data with a distinct feature engineering approach not found in the system's learnwares. While some system learnwares target the PFS dataset, their feature spaces don't match user tasks perfectly. In this scenario, we assess various data-free reusers, and the results on the left side of Fig. 5 show that the system performs well even when users lack labeled data, especially with the AverageEnsemble method.

Different task scenarios. We use three different feature engineering methods for ten stores in the M5 dataset, creating 30 users. While there are learnwares in the system designed for sales forecasting task, none of them are specifically designed to meet the M5 requirements. On the right side of Fig. 5, we show loss curves for the user's self-trained model and various learnware reuse methods, where heterogeneous learnwares prove effective with limited labeled data for better alignment with the user's specific task.

6.2 Experiments on Image and Text Data

Currently, our solutions are generally designed for any type of data and models, thus for unstructured data like image and text scenarios, we only assess our system on classic benchmark data. Images of varying sizes can be standardized by resizing, and text data was represented through a sentence embedding extractor.

Image experiments. Based on the image classification dataset CIFAR-10 [11], we upload 50 learnwares, each containing a convolutional neural network trained on an unbalanced subset of 12,000 samples from four categories, with a sampling ratio of 0.4 : 0.4 : 0.1 : 0.1. In this scenario, we evaluate 100 user tasks, each consisting of 3,000 CIFAR-10 samples across six categories, with a sampling ratio of 0.3 : 0.3 : 0.1 : 0.1 : 0.1 : 0.1. The performance is assessed using 1 - Accuracy as the loss metric. Fig. 6 shows that when users face a scarcity of labeled data or possess only a limited amount of it (less than 2,000 instances), leveraging existing learnwares can yield good performances.

Method	Loss
Mean in System	0.493
Best in System	0.141
Top-1	0.154
JobSelector	0.155
AverageEnsemble	0.138

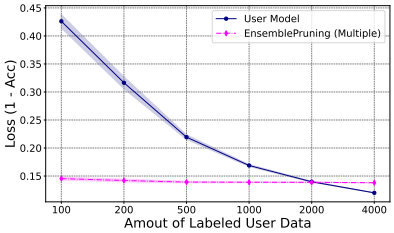

Figure 7: Results of text experiments.

Text Experiments. We use the 20-newsgroup text classification dataset [8], containing roughly 20,000 documents distributed among 20 newsgroups. Similar to the image experiments, 50 text learnwares are uploaded into Beimingwu, with each learnware's model trained on a subset comprising half the samples from three superclasses, utilizing a tf-idf feature extractor combined with a naive Bayes classifier. We define 10 user tasks, and each of them encompasses two superclasses. Figure 7 shows that even without any labeled data, learnware search and reuse can achieve performance comparable to the best learnware in the system. Furthermore, utilizing the learnware dock system results in a reduction of approximately 2,000 samples compared to training models from scratch.

6.3 Real-World Applications

After internal testing, as a research platform for learnware studies, Beimingwu has been open to the academic community, and has been registered by over 500 researchers from more than 150 universities.

Besides, as shown in the system document [5], we demonstrate the effectiveness of Beimingwu in an industrial scenario. A well-trained pump frequency control model can help a pump achieve lower energy consumption. But traditionally, training such a model from scratch is often challenging due to the scarcity of sufficient operational data. By collecting 10 related historical learnwares in Beimingwu, it becomes possible to solve new tasks, whereas identifying which model to deploy is still challenging due to the high cost of trials and data privacy concerns. By utilizing Beimingwu to automatically identify suitable models without leaking raw data, the average power consumption per 1,000 tons of water was reduced from 35.1 kWh to 31.0 kWh, resulting in an 11.7% overall decrease while maintaining consistent pump outlet flow.

7 RELATED WORK

Instead of building models from scratch, for the first time, the learnware paradigm [40, 41] proposes to build a large model platform consisting of numerous high-performing models, and enable users to easily leverage existing models to solve their tasks. Recently, utilizing the large learnware model platform to solve new learning tasks has witnessed a rapidly increasing attention, notably the Hugging Face platform, hosting over half a million models. As git-based remote hosting platforms, they manages learning models like code, with semantic descriptions capturing model information. Fundamentally differing from passive hosting, based on a novel specification-based architecture, Beimingwu aims to automatically identify and assemble high-performing models suitable for user tasks, with no need for extensive data and expert knowledge, while preserving raw data. To achieve this, with machine learning models being functions from the input domain to the output domain, statistical information is necessary to capture their implicit capabilities. Without statistical specifications, it would be forced to examine all potentially helpful models and their combinations on user data, which is unacceptable in terms of computational cost and privacy.

Besides, there has been research aiming to generally reduce the technical burden for model developers to deploy a shared model locally, like Infaas [22] and Acumos [38], whereas they doesn't consider the process of model identification.

Recently, there is a series of works aiming to adopt popular large language models (LLMs) [3] to identify helpful models based on matching the natural language descriptions of each model in the platform and the user's requirements, like HuggingGPT [24], Chameleon [15], and Gorilla [21]. However, in many cases accurate model characterization and identification can not be realized without statistical specification. Furthermore, statistical specification can enable models to be used beyond their original purposes. There are also studies about assessing the reusability or transferability of pre-trained models [4, 17, 34, 36], whereas they generally make the forward pass for all models on user data without considering data privacy. Besides, it is unaffordable to access their combinations on user data due to combinatorial explosion.

Domain adaptation [1] and transfer learning [20] aim to transfer knowledge from a source domain to a target domain, and assumes that the raw source data is available [19, 42]. To relax the requirements for source data, several topics focus on adapting the models from source domain to the target domain, like source-free domain adaptation [33], hypothesis transfer learning [12], model reuse [37], domain adaptation with auxiliary classifiers [5, 13], etc. The learnware paradigm highly differs from these fields because it aims to identify and assemble helpful models from numerous models in a unified and data-preserving way to solve new tasks.

8 CONCLUSION

In this paper, we present Beimingwu: the first open-source learnware dock system providing foundational support for learnware research. The system engine & research toolkit, and the system frontend & backend have been released. The scalable architecture and implementations enable the system to continuously expand its knowledge base and improve its capabilities through the constant submission of learnwares and advancements in algorithmic research, and this continuous evolution of learnware dock system equips it with lifelong learning capability to tackle more varied user tasks. We expect the research community to benefit from Beimingwu for learnware-related algorithm and system studies.

ACKNOWLEDGMENTS

We appreciate the support of Polixir team in various aspects like system deployment and industrial applications. We appreciate the help of Hao-Yu Shi and Xin-Yu Zhang in image and text scenarios, and Lan-Zhe Guo, Zi-Xuan Chen, Zhi Zhou and Yi-Xuan Jin for their help in the early-stage prototype. We thank LAMDA members for their contributions in the system's internal testing phase.

[5] https://docs.bmwu.cloud/

REFERENCES

[1] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. 2006. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19*.

[2] Alain Berlinet and Christine Thomas-Agnan. 2011. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*. 1877–1901.

[4] Yao-Xiang Ding, Xi-Zhu Wu, Kun Zhou, and Zhi-Hua Zhou. 2022. Pre-trained model reusability evaluation for small-data transfer learning. In *Advances in Neural Information Processing Systems 35*. 37389–37400.

[5] Lixin Duan, Ivor W Tsang, Dong Xu, and Tat-Seng Chua. 2009. Domain adaptation from multiple sources via auxiliary classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 289–296.

[6] Lan-Zhe Guo, Zhi Zhou, Yu-Feng Li, and Zhi-Hua Zhou. 2023. Identifying useful learnwares for heterogeneous label spaces. In *Proceedings of the 40th International Conference on Machine Learning*. 12122–12131.

[7] Arthur Jacot, Clément Hongler, and Franck Gabriel. 2018. Neural tangent kernel: convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*. 8580–8589.

[8] Thorsten Joachims. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of the 14th International Conference on Machine Learning*. 143–151.

[9] Kaggle. 2017. Corporacion favorita grocery sales forecasting. https://www.kaggle.com/c/favorita-grocery-sales-forecasting. Accessed: 2023-06-20.

[10] Kaggle. 2018. Predict future sales. https://kaggle.com/competitions/competitive-data-science-predict-future-sales. Accessed: 2023-05-20.

[11] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[12] Ilja Kuzborskij and Francesco Orabona. 2013. Stability and hypothesis transfer learning. In *Proceedings of the 30th International Conference on Machine Learning*. 942–950.

[13] Nan Li, Ivor W Tsang, and Zhi-Hua Zhou. 2013. Efficient optimization of performance measures by classifier adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 6 (2013), 1370–1382.

[14] Jian-Dong Liu, Zhi-Hao Tan, and Zhi-Hua Zhou. 2024. Towards making learnware specification and market evolvable. In *Proceedings of the 38th AAAI Conference on Artificial Intelligence*. 13909–13917.

[15] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems 36*. 43447–43478.

[16] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2022. The M5 competition: background, organization, and implementation. *International Journal of Forecasting* 38, 4 (2022), 1325–1336.

[17] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. 2020. Leep: A new measure to evaluate transferability of learned representations. In *Proceedings of the 37th International Conference on Machine Learning*. 7294–7305.

[18] Roman Novak, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. 2022. Fast finite width neural tangent kernel. In *Proceedings of the 39th International Conference on Machine Learning*. 17018–17044.

[19] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. 2011. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks* 22, 2 (2011), 199–210.

[20] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359.

[21] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334* (2023).

[22] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference*. 397–411.

[23] Bernhard Scholkopf, Sebastian Mika, Chris JC Burges, Philipp Knirsch, K-R Muller, Gunnar Ratsch, and Alexander J Smola. 1999. Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks* 10, 5 (1999), 1000–1017.

[24] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. In *Advances in Neural Information Processing Systems 36*. 38154–38180.

[25] Peng Tan, Zhi-Hao Tan, Yuan Jiang, and Zhi-Hua Zhou. 2023. Handling learnwares developed from heterogeneous feature spaces without auxiliary data. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence*. 4235–4243.

[26] Peng Tan, Zhi-Hao Tan, Yuan Jiang, and Zhi-Hua Zhou. 2024. Towards enabling learnware to handle heterogeneous feature spaces. *Machine Learning* 113, 4 (2024), 1839–1860.

[27] Zhi-Hao Tan, Hao-Yu Shi, Zi-Xuan Chen, and Jiang Yuan. 2024. Learnware reduced kernel mean embedding specification based on neural tangent kernel. *Chinese Journal of Computers* 47, 6 (2024), 1232–1243.

[28] Zhi-Hao Tan, Yi Xie, Yuan Jiang, and Zhi-Hua Zhou. 2022. Real-valued backpropagation is unsuitable for complex-valued neural networks. In *Advances in Neural Information Processing Systems 35*. 34052–34063.

[29] Zifeng Wang and Jimeng Sun. 2022. Transtab: Learning transferable tabular transformers across tables. In *Advances in Neural Information Processing Systems 35*. 2902–2915.

[30] Xi-Zhu Wu, Wenkai Xu, Song Liu, and Zhi-Hua Zhou. 2023. Model reuse with reduced kernel mean embedding specification. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (2023), 699–710.

[31] Yu-Chang Wu, Yi-Xiao He, Chao Qian, and Zhi-Hua Zhou. 2022. Multi-objective evolutionary ensemble pruning guided by margin distribution. In *Proceedings of the 17th International Conference on Parallel Problem Solving from Nature*. 427–441.

[32] Yi Xie, Zhi-Hao Tan, Yuan Jiang, and Zhi-Hua Zhou. 2023. Identifying helpful learnwares without examining the whole market. In *Proceedings of the 26th European Conference on Artificial Intelligence*. 2752–2759.

[33] Shiqi Yang, Yaxing Wang, Kai Wang, Shangling Jui, and Joost van de Weijer. 2022. Attracting and dispersing: A simple approach for source-free domain adaptation. In *Advances in Neural Information Processing Systems 35*. 5802–5815.

[34] Kaichao You, Yong Liu, Ziyang Zhang, Jianmin Wang, Michael I Jordan, and Mingsheng Long. 2022. Ranking and tuning pre-trained models: a new paradigm for exploiting model hubs. *Journal of Machine Learning Research* 23, 209 (2022), 1–47.

[35] Yu-Jie Zhang, Yu-Hu Yan, Peng Zhao, and Zhi-Hua Zhou. 2021. Towards enabling learnware to handle unseen jobs. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*. 10964–10972.

[36] Yi-Kai Zhang, Ting-Ji Huang, Yao-Xiang Ding, De-Chuan Zhan, and Han-Jia Ye. 2023. Model spider: learning to rank pre-trained models efficiently. In *Advances in Neural Information Processing Systems 36*. 13692–13719.

[37] Peng Zhao, Le-Wen Cai, and Zhi-Hua Zhou. 2020. Handling concept drift via model reuse. *Machine Learning* 109 (2020), 533–568.

[38] Shuai Zhao, Manoop Talasila, Guy Jacobson, Cristian Borcea, Syed Anwar Aftab, and John F Murray. 2018. Packaging and sharing machine learning models via the acumos ai open platform. In *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications*. 841–846.

[39] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. CRC press.

[40] Zhi-Hua Zhou. 2016. Learnware: on the future of machine learning. *Frontiers of Computer Science* 10 (2016), 589–590.

[41] Zhi-Hua Zhou and Zhi-Hao Tan. 2024. Learnware: small models do big. *Science China Information Sciences* 67, 1 (2024), 112102.

[42] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2021. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2021), 43–76.