Dynamic Learnware Filtering for Efficient Learnware Identification and System Slimming

Jian-Dong Liu
National Key Laboratory for Novel Software Technology,
School of Artificial Intelligence,
Nanjing University
Nanjing, China
liujd@lamda.nju.edu.cn

Zhi-Hao Tan
National Key Laboratory for Novel Software Technology,
School of Artificial Intelligence,
Nanjing University
Nanjing, China
tanzh@lamda.nju.edu.cn

Zhi-Hua Zhou*
National Key Laboratory for Novel Software Technology,
School of Artificial Intelligence,
Nanjing University
Nanjing, China
zhouzh@lamda.nju.edu.cn

Abstract

The *learnware paradigm* proposed by Zhou [34] aims to solve machine learning tasks by leveraging numerous existing high-performing models instead of training from scratch. These models are accommodated in a *learnware dock system*, where each learnware consists of a model and a *specification* that characterizes the model's utility, enabling it to be identified for future tasks. A critical challenge in this paradigm remains unresolved: determining what models can be or should be admitted to the system. Without well-established admission criteria, the uncontrolled growth of uploaded models could lead to significant redundancy and inefficiency, resulting in higher storage overhead, increased computational costs, and even potential system failure. To address this gap, this paper presents the first attempt to establish learnware admission criteria and dynamically filter redundant learnwares based on model capability coverage. Specifically, we organize task information from all learnwares into a tree-based structure to assess model capabilities across a continuously expanding task set. Using this structure and model capability representation, we develop an efficient and scalable method for detecting redundant learnwares dynamically without traversing the entire system. Theoretical analysis and extensive experiments involving over ten thousand simulated learnwares validate the efficacy and efficiency of our approach.

CCS Concepts

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

Keywords

Machine Learning, Learnware, Learnware Dock System, Learnware Specification, Dynamic Learnware Filtering, System Slimming

*Corresponding author

KDD '25, Toronto, ON, Canada.

ACM ISBN 979-8-4007-1454-2/25/08
https://doi.org/10.1145/3711896.3736917

ACM Reference Format:
Jian-Dong Liu, Zhi-Hao Tan, and Zhi-Hua Zhou. 2025. Dynamic Learnware Filtering for Efficient Learnware Identification and System Slimming. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3–7, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3711896.3736917

1 Introduction

The *learnware paradigm* [34, 35] was proposed to enable users to solve new tasks by reusing existing models instead of starting from scratch. "Learnware = Model + Specification", i.e., a learnware consists of a well-performing model of any structure and a *specification* that captures the model's utility and characteristics, such as its statistical properties. Developers worldwide can submit models trained on various tasks to a *learnware dock system*, which helps generate specifications at developer side to form learnwares; briefly speaking, the learnware dock system will send some "key" information, such as a mathematical function and the size of the specification, to the developer. Then, based on this information and the raw training data, the developer will generate the specification to submit together with the model, without disclosing the raw training data to the learnware dock system. When faced with a new user task, where the user submits her request like a specification, the system automatically identifies and assembles helpful learnwares with the help of specifications. The user can then apply these learnwares directly or refine them with her own data to address the task. Importantly, the system does not access the raw data of either model developers or users.

The learnware paradigm offers substantial advantages in addressing common machine learning challenges. It empowers users to achieve high-quality models by leveraging pre-existing, well-performing learnwares, even if they possess limited data or machine learning expertise, thereby streamlining the model development process. This paradigm inherently supports continual learning and prevents catastrophic forgetting, as the learnware dock system's capability is constantly enriched with new model submissions from diverse tasks, allowing the system to evolve and improve over time. Furthermore, the paradigm is designed to protect data privacy as neither model developers nor users need to disclose their raw data to the learnware dock system or to each other. Recent research has significantly advanced this framework. Notably, the reduced kernel mean embedding (RKME) specification [35] is proposed to identify learnwares by matching their original data distributions with user tasks, which has been demonstrated effective in various

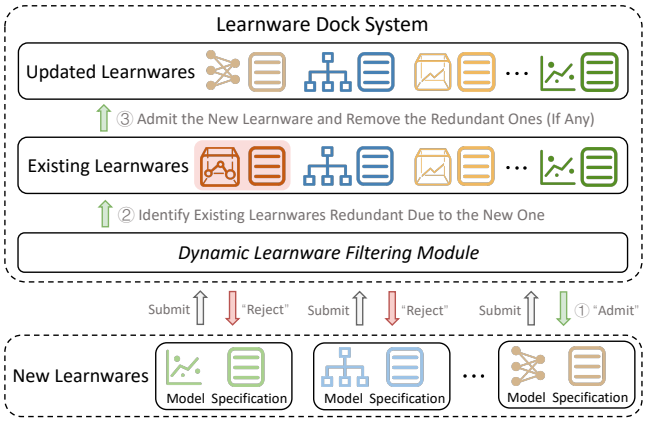

Figure 1: The workflow of dynamic learnware filtering.

tasks [25, 28] and is theoretically proven to protect developers' raw data [15]. Building on this, subsequent research has extended to efficient identification [16, 29], heterogeneous feature spaces [23, 24], and heterogeneous label spaces [10]. Based on the above works, the first learnware dock system, Beimingwu [25], was recently released, serving as a research platform and providing implementations for the entire process of learnware paradigm, with further work extending its application to language models [26].

To realize the vision depicted above, a critical problem in the learnware paradigm is determining *what models can be or should be admitted to the learnware dock system*. Imagine that thousands to millions of models will be shared by developers for future reuse. Admitting models unrestrictedly could result in numerous redundant models, significantly increasing storage and computational costs for management and retrieval. This slows response times, increases latency, and reduces throughput, ultimately impairing system performance and potentially causing system failure. Furthermore, malicious attackers could submit excessive redundant models to exhaust system resources, leading to a denial-of-service attack. Given the recent establishment of Beimingwu system [25], addressing this problem has become even more urgent.

Although this issue is crucial, it remains unexplored in existing learnware literature. Moreover, it presents several key fundamental challenges. First, detecting redundant learnwares requires characterizing capability differences between models, which is quite non-trivial due to the inherent intricacy of model functional spaces and the inaccessibility of raw data. Second, even when models differ, they may still be redundant; the lack of a clear definition of redundant learnwares further complicates this problem. Third, filtering redundant learnwares is an inherently dynamic process, as previously admitted learnwares may become redundant due to the system's continuous expansion with newly submitted learnwares. Furthermore, this dynamic filtering process must be both efficient and scalable to handle the growing number of learnwares submitted by developers worldwide.

To address these challenges, our key insight is to characterize model capabilities across a continuously expanding and structured set of tasks and establish criteria for filtering redundant learnwares based on capability coverage. Specifically, the RKME specification incorporates the original task information of each model, enabling us to structurally organize task information from existing learnwares for evaluating model capabilities without accessing raw data. As the task set grows with the continuous expansion of the learnware dock system, this evolving evaluation of model capabilities will become increasingly accurate and comprehensive. Using this capability representation, we can establish criteria for dynamically filtering redundant learnwares by comparing their capabilities and identifying those already covered by existing learnwares, as shown in Figure 1. Our main contributions are summarized as follows:

- We present the *first* attempt to establish learnware admission criteria and dynamically filter redundant learnwares from the perspective of model capability coverage, without accessing original data. Our proposed approach reduces system storage overhead, enhances system management and search efficiency, while also defending against malicious attacks involving excessive uploads of redundant learnwares.
- To ensure efficiency and scalability of our approach, we structurally organize task information from all learnwares into an RKME cover tree, enabling model capability evaluation across a continuously expanding task set. Based on this structure, we propose an efficient method for dynamically filtering redundant learnwares without traversing the entire system.
- Theoretical analysis, extensive experiments with over ten thousand models across real-world scenarios, and an ablation study validate the effectiveness and efficiency of our approach in reducing system size and improving identification efficiency while preserving system performance.

2 Preliminary

As the specification plays a central role in the learnware paradigm, this section introduces the recently proposed reduced kernel mean embedding (RKME) specification [35], which has shown its efficacy in several learnware studies [16, 23, 28] by concisely representing the model's training data distribution with kernel methods.

We begin by introducing the kernel mean embedding (KME) [22], which maps a probability distribution to a point in reproducing kernel Hilbert space (RKHS) and effectively captures distribution information. For a distribution $\mathcal{D}$ over $\mathcal{X}$ and a kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ with RKHS $\mathcal{H}_k$, the KME is defined as $\mu_{\mathcal{D}} = \int_{\mathcal{X}} k(\boldsymbol{x}, \cdot) \mathrm{d}\mathcal{D}(\boldsymbol{x})$, which is approximated by the empirical KME $\hat{\mu}_{\mathcal{D}} = \frac{1}{m} \sum_{i=1}^{m} k(\boldsymbol{x}_i, \cdot)$ in practice, where the samples $\{\boldsymbol{x}_i\}_{i=1}^{m}$ are drawn i.i.d. from $\mathcal{D}$. Under mild conditions, $\hat{\mu}_{\mathcal{D}}$ converges to $\mu_{\mathcal{D}}$ at a rate of $O(1/\sqrt{m})$, measured by the RKHS norm $\|\cdot\|_{\mathcal{H}_k}$ [22].

While KME requires access to raw data, which conflicts with the principle of data privacy in the learnware paradigm, the RKME specification is proposed to retain the advantages of KME while protecting data privacy. Specifically, the RKME specification serves as a reduced set $\{(\beta_j \in \mathbb{R}, \boldsymbol{z}_j \in \mathcal{X})\}_{j=1}^{n}$, which approximates the empirical KME $\hat{\mu}_{\mathcal{D}}$ by solving:

$$\min_{\boldsymbol{\beta}, \boldsymbol{z}} \left\| \frac{1}{m} \sum_{i=1}^{m} k(\boldsymbol{x}_i, \cdot) - \sum_{j=1}^{n} \beta_j k(\boldsymbol{z}_j, \cdot) \right\|_{\mathcal{H}_k}^2, \tag{1}$$

in an alternating optimization manner [28], with the RKME $\tilde{\mu}_{\mathcal{D}} = \sum_{j=1}^{n} \beta_j k(\boldsymbol{z}_j, \cdot)$ converging to $\hat{\mu}_{\mathcal{D}}$ at rate $O(1/\sqrt{n})$ [1, 33]. Moreover, this specification is proven to scarcely contain any original data and possesses robust defense against common inference attacks [15].

3 Problem Formulation

This work focuses on dynamically detecting whether a model should be admitted and retained by the learnware dock system, primarily centering on the submitting stage of the learnware paradigm. At this stage, developers worldwide continuously submit their well-performing models along with corresponding RKME specifications. Assume there are N existing learnwares in the system. The developer of the i-th learnware has access to a private local dataset $D_i = (\boldsymbol{X}_i, \boldsymbol{y}_i)$, where instances are sampled from a distribution $\mathcal{D}_i$ over the input space $\mathcal{X}$, and labels are determined by a ground-truth function $h \in \mathcal{F} = \{f \mid f : \mathcal{X} \mapsto \mathcal{Y}\}$, i.e.,

$$\forall (\boldsymbol{x}, y) \in D_i, \boldsymbol{x} \sim \mathcal{D}_i, y = h(\boldsymbol{x}).$$

Using the local dataset D_i, the developer trains a high-performing model $f_i \in \mathcal{F}$ that satisfies the following property:

$$\mathcal{L}_{\mathcal{D}_i}(f_i, h) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_i}[\ell(f_i(\boldsymbol{x}), h(\boldsymbol{x}))] \leq \varepsilon, \tag{2}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ is a symmetric loss function. To approximate the i-th task without exposing raw data, an RKME specification $R_i = (\boldsymbol{\beta}_i \in \Delta^n, \boldsymbol{Z}_i \in \mathcal{X}^n)$ is constructed, where Δ^n denotes the n-dimensional simplex. The developer then submits the learnware (f_i, R_i) to the learnware dock system.

Given N existing learnwares $\mathcal{S} = \{(f_i, R_i)\}_{i=1}^N$ in the system, the task is to determine whether a new learnware $(f_{\text{new}}, R_{\text{new}})$ should be admitted. If admitted, the system is required to identify any existing learnwares that become redundant due to the new learnware and filter them out.

4 Proposed Approach

To detect whether a learnware is redundant and should not be admitted or retained by the system, a straightforward approach is to compare its training tasks with those of other learnwares. However, since machine learning models are essentially functions in a functional space, their true capabilities extend beyond their training tasks, leading to inaccurate detection results. To improve detection accuracy, we propose dynamically evaluating a learnware's capabilities on a continuously expanding set of tasks. Inspired by the evolvable learnware specification [16], this can be achieved by utilizing the task information within RKME specifications from existing learnwares. Considering a new learnware $(f_{\text{new}}, R_{\text{new}})$ and a task set $\{\mathcal{D}_i\}_{i=1}^N$ derived from existing learnwares, the generalization error of the model f_{new} on the i-th task is defined as follows:

$$\mathcal{L}_{\mathcal{D}_i}(f_{\text{new}}, h) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_i}[\ell(f_{\text{new}}(\boldsymbol{x}), h(\boldsymbol{x}))].$$

Since the original distribution $\mathcal{D}_i$ and the ground-truth function h are inaccessible, we approximate them using the RKME specification $R_i = (\boldsymbol{\beta}_i, \boldsymbol{Z}_i)$ and the high-performing model f_i, respectively. To estimate the generalization error, we then use the empirical loss on the RKME specification R_i, defined as follows:

$$\widetilde{\mathcal{L}}_{R_i}(f_{\text{new}}, f_i) = \sum\nolimits_{j=1}^{n} \beta_{i,j}\ell(f_{\text{new}}(\boldsymbol{z}_{i,j}), f_i(\boldsymbol{z}_{i,j})).$$

The difference between $\widetilde{\mathcal{L}}_{R_i}(f_{\text{new}}, f_i)$ and $\mathcal{L}_{\mathcal{D}_i}(f_{\text{new}}, h)$ is bounded by the following theorem, with the proof provided in Appendix B.1.

Theorem 4.1. *Assume* $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) < \infty$ *and that the loss function* ℓ *satisfies the triangle inequality. Let* $\ell_{f,f'} : \boldsymbol{x} \mapsto \ell(f(\boldsymbol{x}), f'(\boldsymbol{x}))$, *and assume* $\|\ell_{f,f'}\|_{\mathcal{H}_k}$ *is uniformly bounded for all* $f, f' \in \mathcal{F}$. *Then, with probability at least* $1 - \delta(\delta \in (0, 1))$, *for all* $f \in \mathcal{F}$, *we have:*

$$\left|\mathcal{L}_{\mathcal{D}_i}(f, h) - \widetilde{\mathcal{L}}_{R_i}(f, f_i)\right| = O\left(\varepsilon + m^{-\frac{1}{2}} + n^{-\frac{1}{2}}\right),$$

where ε *is defined in Eq.* (2), *with* m *and* n *representing the sizes of the training dataset* D_i *and RKME specification* R_i, *respectively.*

Based on above approximation, for any model f, we can leverage numerous existing learnwares $\mathcal{S} = \{(f_i, R_i)\}_{i=1}^N$ to determine the task set that f excels at, as follows:

$$\mathcal{I}_f(\mathcal{S}) = \left\{R_i \mid \widetilde{\mathcal{L}}_{R_i}(f, f_i) \leq \xi, (f_i, R_i) \in \mathcal{S}\right\},$$

where ξ is a performance threshold. With the continuous expansion of $\mathcal{S}$ (i.e., the learnware dock system continuously scales up), $\mathcal{I}_f(\mathcal{S})$ offers an increasingly precise characterization for the capabilities of model f. Using this representation, we establish the admission criteria for identifying redundant learnwares.

Criteria 4.2. Given existing learnwares $\mathcal{S} = \{(f_i, R_i)\}_{i=1}^N$ and a performance threshold ξ, a new learnware $(f_{\text{new}}, R_{\text{new}})$ is considered redundant and should be rejected if some $(f_i, R_i) \in \mathcal{S}$ satisfies:

$$\mathcal{I}_{f_{\text{new}}}(\mathcal{S}) \subseteq \mathcal{I}_{f_i}(\mathcal{S}), \tag{3}$$

$$\forall R \in \mathcal{I}_{f_{\text{new}}}(\mathcal{S}) \cup \{R_{\text{new}}\}, \widetilde{\mathcal{L}}_R(f_{\text{new}}, f_i) \leq \xi. \tag{4}$$

The first condition in Eq. (3) states that the capabilities of f_{new} are covered by an existing learnware, as f_i excels at all tasks in $\mathcal{S}$ that f_{new} masters. Since this condition only focuses on overall model performance, the two may perform differently on individual samples. Thus, the second condition in Eq. (4) further ensures that where f_{new} performs well, including its original task indicated by R_{new}, f_i not only achieves similar performance but also produces highly consistent outputs, allowing the new learnware $(f_{\text{new}}, R_{\text{new}})$ to be replaced by the existing learnware. Since the representation $\mathcal{I}_f(\mathcal{S})$ evolves with the expansion of $\mathcal{S}$, the above conditions lead to increasing detection accuracy as the system scales up.

Although the above criteria are sufficient for detecting redundant learnwares, the detection process is still challenging:

- A brute-force check for the redundancy of $(f_{\text{new}}, R_{\text{new}})$ requires traversing all existing learnwares. Since assessing the first condition in Eq. (3) is potentially $O(N)$, the worst-case time complexity is $O(N^2)$, which is unacceptable.
- If $(f_{\text{new}}, R_{\text{new}})$ is not redundant, we need to identify the existing learnwares that become redundant due to $(f_{\text{new}}, R_{\text{new}})$. This process also necessitates traversing all learnwares, with the same worst-case time complexity of $O(N^2)$.

To address these challenges and avoid traversing all learnwares, we propose an efficient and scalable approach for dynamically filtering redundant learnwares. In the following subsections, we start with the structural organization of learnwares, then introduce the dynamic filtering method for redundant learnwares. For clarity, the primary notations used are summarized in Appendix A.

4.1 Structural Organization of Learnwares

For efficient filtering of redundant learnwares, it is crucial to compute $\mathcal{I}_f(\mathcal{S})$ efficiently without traversing the entire system. To achieve this, our key observation is that for any model f, the difference between $\widetilde{\mathcal{L}}_{R_i}(f, f_i)$ and $\widetilde{\mathcal{L}}_{R_j}(f, f_j)$ is related to the distance

Algorithm 1: RKME Cover Tree Insertion

Input: RKME cover tree $\mathcal{T}(\mathcal{R})$, a new RKME specification p.
Output: The updated tree $\mathcal{T}(\mathcal{R} \cup \{p\})$.

Function *Insert(tree $\mathcal{T}(\mathcal{R})$, node p)*:
  Let r be the root of $\mathcal{T}(\mathcal{R})$ and set $i \leftarrow \max_{q \in \mathcal{R} \setminus \{r\}} l(q)$;
  if *_Insert(i, $\{r\}$, $\mathcal{T}(\mathcal{R})$, p)* = `False` **then**
    Set $l(p) \leftarrow \lceil \log_2 d_{\mathcal{H}_k}(p, r) \rceil - 1$;
    Insert p into Children(r);
    Set $l(r) \leftarrow \max(l(r), l(p) + 1)$;
  end
  return *The updated tree $\mathcal{T}(\mathcal{R} \cup \{p\})$*
end
Function *_Insert(level i, node set Q, tree $\mathcal{T}(\mathcal{R})$, node p)*:
  if $i < \min_{q \in \mathcal{R}} l(q)$ **or** $Q = \emptyset$ **then return** `False`;
  $V \leftarrow \cup_{q \in Q} \{a \in \text{Children}(q) \mid l(a) = i\}$;
  $Q_i \leftarrow \{a \in (V \cup Q) \mid d_{\mathcal{H}_k}(a, p) \le 2^{i+1}\}$;
  Set next level $t \leftarrow \max_{a \in Q_i} \text{Next}(a, i)$;
  if *_Insert(t, Q_i, $\mathcal{T}(\mathcal{R})$, p)* = `False` **then**
    if $\min_{q \in Q} d_{\mathcal{H}_k}(p, q) > 2^i$ **then return** `False`;
    Pick $v \in Q_i$ minimizing $d_{\mathcal{H}_k}(p, q)$;
    Set $l(p) \leftarrow \lceil \log_2 d_{\mathcal{H}_k}(p, v) \rceil - 1$;
    Insert p into Children(v);
  end
  return `True`
end

between the specifications R_i and R_j. Let $\tilde{\mu}_{\mathcal{D}_i}$ denote the i-th RKME for the specification R_i, defined as

$$\tilde{\mu}_{\mathcal{D}_i} = \sum_{j=1}^{n} \beta_{i,j} k(\boldsymbol{z}_{i,j}, \cdot).$$

The distance metric between any two RKME specifications, R_i and R_j, is derived within the associated RKHS $\mathcal{H}_k$ as follows:

$$d_{\mathcal{H}_k}(R_i, R_j) = \left\| \tilde{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_j} \right\|_{\mathcal{H}_k}.$$

Utilizing this distance metric, we formalize the relationship between the loss difference and the specification distance in the following theorem, with the proof provided in Appendix B.2.

Proposition 4.3. *With the assumptions from Theorem 4.1, let (f_i, R_i) and (f_j, R_j) be two learnwares. Then, for all $f \in \mathcal{F}$, we have:*

$$\left| \widetilde{\mathcal{L}}_{R_i}(f, f_i) - \widetilde{\mathcal{L}}_{R_j}(f, f_j) \right| = O_{\mathbb{P}}\left(d_{\mathcal{H}_k}(R_i, R_j) + \eta \right),$$

where $\eta = \varepsilon + m^{-\frac{1}{2}} + n^{-\frac{1}{2}}$ with ε, m, n defined in Theorem 4.1.

Proposition 4.3 indicates that the difference in loss can be approximately represented by the specification distance $d_{\mathcal{H}_k}(R_i, R_j)$. Specifically, when $\widetilde{\mathcal{L}}_{R_i}(f, f_i)$ is computed, we can estimate that $\widetilde{\mathcal{L}}_{R_j}(f, f_j)$ lies within the interval $[\widetilde{\mathcal{L}}_{R_i}(f, f_i) - w \cdot d_{\mathcal{H}_k}(R_i, R_j), \widetilde{\mathcal{L}}_{R_i}(f, f_i) + w \cdot d_{\mathcal{H}_k}(R_i, R_j)]$, where w is a weighting constant. If the left endpoint of this interval exceeds the performance threshold ξ, the computation for $\widetilde{\mathcal{L}}_{R_j}(f, f_j)$ can be skipped, thereby avoiding the need to traverse all learnwares and accelerating the calculation of $\mathcal{I}_f(\mathcal{S})$.

RKME Cover Tree. Based on the above observation, we propose hierarchically organizing learnwares using the distance metric

Algorithm 2: Calculation of the task set $\mathcal{I}_f(\mathcal{S})$

Input: RKME cover tree $\mathcal{T}(\mathcal{R})$, a model f, constants ξ, w.
Output: The task set $\mathcal{I}_f(\mathcal{S})$ that f excels at.

1 Let r be the root of $\mathcal{T}(\mathcal{R})$ and set $Q \leftarrow \{r\}, \mathcal{I}_f \leftarrow \emptyset$;
2 **while** $Q \neq \emptyset$ **do**
3 Pick $p \in Q$ and set $Q \leftarrow Q \setminus \{p\}$;
4 Let (f_p, R_p) be the learnware associated with the node p;
5 **if** $\widetilde{\mathcal{L}}_{R_p}(f, f_p) \le \xi$ **then** $\mathcal{I}_f \leftarrow \mathcal{I}_f \cup \{R_p\}$;
6 **if** $\widetilde{\mathcal{L}}_{R_p}(f, f_p) - w \cdot \mathit{Covdist}(p) \le \xi$ **then**
7 Set $Q \leftarrow Q \cup \text{Children}(p)$;
8 **end**
9 **end**
10 **return** *The resulting task set $\mathcal{I}_f(\mathcal{S})$*

of RKMEs, which enables efficient calculation of $\mathcal{I}_f(\mathcal{S})$ without traversing the entire system. Specifically, we construct an RKME cover tree defined in Definition 4.4, leveraging the compressed cover tree [2, 9] as the underlying data structure.

Definition 4.4 (RKME Cover Tree)**.** Let $\mathcal{R}$ be a set of RKME specifications. An RKME cover tree $\mathcal{T}(\mathcal{R})$ has a one-to-one mapping between $\mathcal{R}$ and all nodes of $\mathcal{T}(\mathcal{R})$, with a level function $l : \mathcal{R} \mapsto \mathbb{Z}$ satisfying the following conditions:

- (Root) The root node r satisfies $l(r) \ge 1 + \max_{p \in \mathcal{R} \setminus \{r\}} l(p)$.
- (Covering) Every node $q \in \mathcal{R} \setminus \{r\}$ has a unique parent p such that the distance $d_{\mathcal{H}_k}(q, p) \le 2^{l(q)+1}$ and $l(q) < l(p)$.
- (Separation) For $i \in \mathbb{Z}$, the set $C_i = \{p \in \mathcal{R} \mid l(p) \ge i\}$ satisfies that for any distinct $p, q \in C_i$, $d_{\mathcal{H}_k}(p, q) \ge 2^i$.

For any node $p \in \mathcal{T}(\mathcal{R})$, let Children($p$) and Descendants($p$) denote the set of children and descendants of p, respectively, and define the function $\text{Covdist}(p) = \max_{q \in \text{Descendants}(p)} d_{\mathcal{H}_k}(p, q)$.

Importantly, the RKME cover tree $\mathcal{T}(\mathcal{R})$ supports online construction, allowing for the streaming addition of new nodes to the tree. This property is particularly beneficial for learnware dock systems where learnwares are uploaded in a streaming manner. To explain the insertion process, we define additional notations below.

Definition 4.5 (Children(p, i) and Next(p, i))**.** In an RKME cover tree $\mathcal{T}(\mathcal{R})$, for a node $p \in \mathcal{R}$ and level i, define $\text{Children}(p, i) = \{a \in \text{Children}(p) \mid l(a) = i\}$. Let $\text{Next}(p, i)$ be the largest level in the set $\{j < i \mid \text{Children}(p, j) \neq \emptyset\}$; if it is empty, set $\text{Next}(p, i) = -\infty$.

Based on these definitions, the online construction of the RKME cover tree is presented in Algorithm 1, where the updated tree remains a valid RKME cover tree in Definition 4.4 and the time complexity of this process is $O(\log |\mathcal{R}|)$ with a hidden dimensionality factor. The proofs of these statements are essentially the same as those for the original compressed cover tree [9].

Using the RKME cover tree, the task set $\mathcal{I}_f(\mathcal{S})$ can be obtained through Algorithm 2, enabling efficient computation without examining all tasks in the system. Since the value of Covdist(p) for each node $p \in \mathcal{T}(\mathcal{R})$ can be efficiently precomputed and updated during the insertion process of new nodes, the key operation in line 6 of Algorithm 2 can be executed directly.

4.2 Dynamic Learnware Filtering

Since learnwares are uploaded continuously, our dynamic filtering method is executed upon the arrival of each new learnware. The process involves three steps: (1) check if the new learnware is redundant and reject it if so; (2) if not redundant, identify any existing learnware that becomes redundant due to the new one; (3) admit the new learnware and update the relevant data structures.

Redundancy Detection for New Learnwares. For a new learnware $(f_{\text{new}}, R_{\text{new}})$, we initially compute the task set $\mathcal{I}_{f_{\text{new}}}(\mathcal{S})$ using Algorithm 2, where the learnware set $\mathcal{S}$ is defined in Criteria 4.2. Subsequently, we need to identify existing learnwares whose task sets are supersets of $\mathcal{I}_{f_{\text{new}}}(\mathcal{S})$, defined as:

$$\mathcal{M} = \left\{ f_i \mid \mathcal{I}_{f_{\text{new}}}(\mathcal{S}) \subseteq \mathcal{I}_{f_i}(\mathcal{S}), (f_i, R_i) \in \mathcal{S} \right\}. \tag{5}$$

Since verifying whether $\mathcal{I}_{f_{\text{new}}}(\mathcal{S})$ is a subset of $\mathcal{I}_{f_i}(\mathcal{S})$ incurs a worst-case time complexity of $O(N)$, iterating over all learnwares to compute $\mathcal{M}$ results in a total time complexity of $O(N^2)$, which is impractical. To accelerate the computation of $\mathcal{M}$, we propose assigning a model set $\mathcal{U}_R(\mathcal{S})$ to each specification R of the RKME cover tree $\mathcal{T}(\mathcal{R})$ during its online construction. This set represents the existing models that excel at the task specified by R, defined as:

$$\mathcal{U}_R(\mathcal{S}) = \left\{ f_i \mid R \in \mathcal{I}_{f_i}(\mathcal{S}), (f_i, R_i) \in \mathcal{S} \right\}.$$

In this context, $\mathcal{U}_R(\mathcal{S})$ functions as an inverted index set for $\mathcal{I}_f(\mathcal{S})$, establishing a dual relationship between the two sets. Moreover, the model set $\mathcal{U}_R(\mathcal{S})$ can be efficiently updated during the computation of $\mathcal{I}_f(\mathcal{S})$ in line 5 of Algorithm 2, with total storage overhead consistent with that required for $\mathcal{I}_f(\mathcal{S})$. By leveraging the model set $\mathcal{U}_R(\mathcal{S})$, $\mathcal{M}$ in Eq. (5) can be efficiently computed as:

$$\mathcal{M}' = \bigcap\nolimits_{R \in \mathcal{I}_{f_{\text{new}}}(\mathcal{S})} \mathcal{U}_R(\mathcal{S}). \tag{6}$$

Every element f_i in $\mathcal{M}'$ satisfies the conditions of $\mathcal{M}$ in Eq. (5), and the reverse also holds, leading to the following proposition:

Proposition 4.6. *The model set $\mathcal{M}'$, defined in Eq. (6), is equivalent to the set $\mathcal{M}$, as given in Eq. (5).*

Since $\mathcal{U}_R(\mathcal{S})$ is precomputed for all $R \in \mathcal{I}_{f_{\text{new}}}(\mathcal{S})$, and the set intersection has a worst-case time complexity of $O(N)$, we have:

Proposition 4.7. *The worst-case time complexity of computing $\mathcal{M}'$, as expressed in Eq. (6), is $O(|\mathcal{I}_{f_{new}}(\mathcal{S})|N)$.*

Given that the number of tasks that a model excels at is typically small, i.e., $|\mathcal{I}_{f_{\text{new}}}(\mathcal{S})| \ll N$, the time complexity of computing $\mathcal{M}'$ in Eq. (6) is practical and significantly more efficient than the brute-force computation described in Eq. (5).

If $\mathcal{M}'$ is empty, the new learnware $(f_{\text{new}}, R_{\text{new}})$ is not redundant and should be admitted. Otherwise, we check if any model $f_i \in \mathcal{M}'$ satisfies the second condition in Eq. (4). If such a model exists, the new learnware is rejected; otherwise, it is admitted. The worst-case time complexity of verifying the second condition is $O(|\mathcal{I}_{f_{\text{new}}}(\mathcal{S})| \cdot |\mathcal{M}'| + |\mathcal{M}'|)$. Since $|\mathcal{M}'| \leq N$, the overall worst-case time complexity for this detection remains $O(|\mathcal{I}_{f_{\text{new}}}(\mathcal{S})|N)$.

A special case arises when $\mathcal{I}_{f_{\text{new}}}(\mathcal{S}) = \emptyset$, making $\mathcal{M}$ the set of all models in $\mathcal{S}$. In this case, the task reduces to checking if any model in $\mathcal{M}$ performs well on R_{new}, which is equivalent to determining whether $\mathcal{U}_{R_{\text{new}}}(\mathcal{M})$ is empty, where:

$$\mathcal{U}_{R_{\text{new}}}(\mathcal{M}) = \left\{ f_i \mid \widetilde{\mathcal{L}}_{R_{\text{new}}}(f_i, f_{\text{new}}) \leq \xi, f_i \in \mathcal{M} \right\}, \tag{7}$$

with ξ being the threshold in Criteria 4.2. To compute $\mathcal{U}_{R_{\text{new}}}(\mathcal{M})$ efficiently without traversing all learnwares, we narrow $\mathcal{M}$ via the RKME cover tree $\mathcal{T}(\mathcal{R})$. Based on Proposition 4.3, models performing well on R_{new} likely excel on nearby RKMEs. Thus, we apply a k-nearest neighbor search algorithm [9] to find the K closest RKMEs $\text{KNN}(R_{\text{new}})$, with time complexity $O((K + \log|\mathcal{R}|)\log K)$. Finally, $\mathcal{M}$ is approximated as the union of their model sets, i.e., $\bigcup_{R \in \text{KNN}(R_{\text{new}})} \mathcal{U}_R(\mathcal{S})$, enabling efficient calculation.

Detecting Existing Redundant Learnwares. If the new learnware $(f_{\text{new}}, R_{\text{new}})$ is not redundant, we must determine whether any existing learnware becomes redundant due to the new one. To do so, we identify existing learnwares whose task sets are subsets of the new learnware's set $\mathcal{I}_{f_{\text{new}}}(\mathcal{S})$, i.e.,

$$\mathcal{V} = \left\{ f_i \mid \mathcal{I}_{f_i}(\mathcal{S}) \subseteq \mathcal{I}_{f_{\text{new}}}(\mathcal{S}), (f_i, R_i) \in \mathcal{S} \right\}. \tag{8}$$

To efficiently obtain $\mathcal{V}$ without traversing all learnwares, we reuse the inverted index set $\mathcal{U}_R(\mathcal{S})$. Specifically, let $\text{Count}(f)$ denote the number of tasks in $\mathcal{I}_{f_{\text{new}}}(\mathcal{S})$ where model f excels, defined as:

$$\text{Count}(f) = \left| \left\{ R \in \mathcal{I}_{f_{\text{new}}}(\mathcal{S}) \mid f \in \mathcal{U}_R(\mathcal{S}) \right\} \right|.$$

The model set $\mathcal{V}$ can then be efficiently computed as:

$$\mathcal{V}' = \left\{ f_i \in \bigcup\nolimits_{R \in \mathcal{I}_{f_{\text{new}}}(\mathcal{S})} \mathcal{U}_R(\mathcal{S}) \mid \text{Count}(f_i) = |\mathcal{I}_{f_i}(\mathcal{S})| \right\}. \tag{9}$$

Since $\text{Count}(f_i) = |\mathcal{I}_{f_i}(\mathcal{S})|$ implies $\mathcal{I}_{f_i}(\mathcal{S}) \subseteq \mathcal{I}_{f_{\text{new}}}(\mathcal{S})$, each $f_i \in \mathcal{V}'$ satisfies the conditions of $\mathcal{V}$ in Eq. (8), and vice versa, as stated in the following proposition:

Algorithm 3: Dynamic Learnware Filtering

Input: Existing learnwares $\mathcal{S}$, RKME cover tree $\mathcal{T}(\mathcal{R})$, a new learnware $(f_{\text{new}}, R_{\text{new}})$.
Output: "`Admit`" if the new learnware is admitted, "`Reject`" otherwise.

```
Compute the task set I_{f_new}(S) using Algorithm 2;
if I_{f_new}(S) ≠ ∅ then
    Calculate the model set M' using Eq. (6);
    if ∃f_i ∈ M' satisfying Eq. (4) then return "Reject";
else
    Set M ← ⋃_{R∈KNN(R_new)} U_R(S) via k-NN search [9];
    Calculate U_{R_new}(M) according to Eq. (7);
    if U_{R_new}(M) ≠ ∅ then return "Reject";
end
Calculate the model set V' based on I_{f_new}(S) using Eq. (9);
for existing model f_i ∈ V' do
    if f_i satisfies Eq. (10) then Filter out f_i from the system;
end
Insert R_new into the RKME cover tree T(R) via Algorithm 1;
Set S ← S ∪ {(f_new, R_new)} and update corresponding task sets I_f(S) and model sets U_R(S);
return "Admit"
```

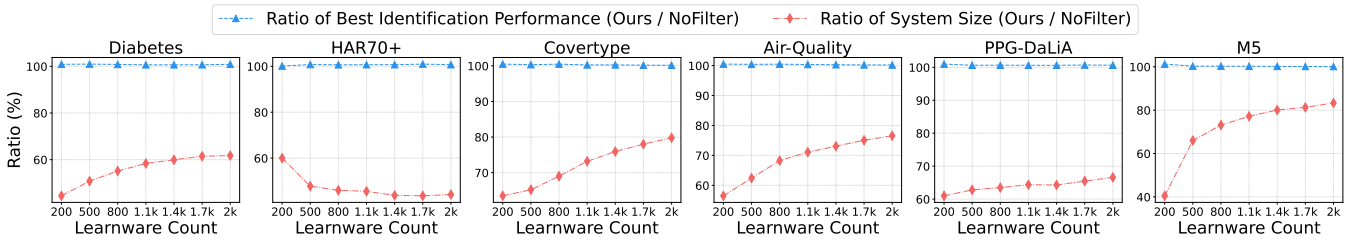

Figure 2: For each scenario, the best identification performance is the minimum loss on the user task among all learnwares in the system, while system size is the total number of accommodated learnwares. The figure shows the ratio (%) of these two metrics under our approach relative to the baseline, as the number of uploaded learnwares continuously increases up to 2,000.

Proposition 4.8. *The model set $\mathcal{V}'$, defined in Eq.* (9)*, is equivalent to the set $\mathcal{V}$, as given in Eq.* (8)*.*

While $|\mathcal{I}_{f_i}(\mathcal{S})|$ is precomputed, $\text{Count}(f)$ and the model set $\mathcal{V}'$ can be computed concurrently based on the multiset union $\biguplus_{R\in\mathcal{I}_{f_{\text{new}}}(\mathcal{S})}\mathcal{U}_R(\mathcal{S})$. Since the worst-case time complexity of the multiset union operation is $O(N)$, the following proposition holds:

Proposition 4.9. *The worst-case time complexity of computing $\mathcal{V}'$, as expressed in Eq.* (9)*, is $O(|\mathcal{I}_{f_{new}}(\mathcal{S})|N)$.*

The time complexity of computing $\mathcal{V}'$ matches that of $\mathcal{M}'$, as both efficiently avoid traversing all learnwares. For each $f_i \in \mathcal{V}'$, we then determine whether it can be replaced by the new one $(f_{\text{new}}, R_{\text{new}})$ by verifying the following condition:

$$\forall R \in \mathcal{I}_{f_i}(\mathcal{S}), \widetilde{\mathcal{L}}_R(f_{\text{new}}, f_i) \leq \xi, \tag{10}$$

with ξ defined in Criteria 4.2. If satisfied, the existing learnware f_i is redundant and removed from the system; otherwise, it is retained. Since $|\mathcal{I}_{f_i}(\mathcal{S})| \leq |\mathcal{I}_{f_{\text{new}}}(\mathcal{S})|$, the time complexity of this process is $O(|\mathcal{I}_{f_{\text{new}}}(\mathcal{S})|\cdot|\mathcal{V}'|)$. Thus, the overall time complexity of detecting redundant existing learnwares remains $O(|\mathcal{I}_{f_{\text{new}}}(\mathcal{S})|N)$, consistent with that of detecting redundant new learnwares.

Admitting Learnwares and Updating Data Structures. If the new learnware $(f_{\text{new}}, R_{\text{new}})$ is not redundant, it is admitted and R_{new} is inserted into the RKME cover tree $\mathcal{T}(\mathcal{R})$ using Algorithm 1. Let $\mathcal{S}' = \mathcal{S} \cup \{(f_{\text{new}}, R_{\text{new}})\}$, $\mathcal{I}_{f_{\text{new}}}(\mathcal{S}') = \mathcal{I}_{f_{\text{new}}}(\mathcal{S}) \cup \{R_{\text{new}}\}$ and $\mathcal{U}_{R_{\text{new}}}(\mathcal{S}') = \mathcal{U}_{R_{\text{new}}}(\mathcal{M}) \cup \{f_{\text{new}}\}$, where $\mathcal{U}_{R_{\text{new}}}(\mathcal{M})$ is computed as described in line 6-7 of Algorithm 3. Other task sets $\mathcal{I}_f(\mathcal{S}')$ and model sets $\mathcal{U}_R(\mathcal{S}')$ are updated accordingly. Since $|\mathcal{U}_{R_{\text{new}}}(\mathcal{S}')|$ and $|\mathcal{I}_{f_{\text{new}}}(\mathcal{S}')|$ are smaller than $|\mathcal{S}'|$, the worst-case time complexity for updating the data structures is $O(N)$, which remains efficient and has a minor impact on the overall computational cost of our approach. The complete process of dynamic learnware filtering for redundant learnwares is summarized in Algorithm 3.

5 Experiments

5.1 Experimental Setup

Here we introduce the experimental setup for scenario construction, evaluation of learnware identification, and configurations. More details on settings and additional results are provided in Appendix C.

Scenario Construction. In our experiments, we develop over 10,000 models of various types, covering diverse real-world classification and regression tasks. These scenarios are associated with six real-world datasets: Diabetes [6], HAR70+ [27], Covertype [3], Air-Quality [31], PPG-DaLiA [20], and M5 [17]. The datasets span a variety of tasks: Diabetes, HAR70+, and Covertype classify diabetes health indicators, human activities, and forest cover types, respectively, while Air-Quality, PPG-DaLiA, and M5 predict air pollutants, heart rate, and product sales, respectively. These datasets vary significantly in size, with the number of instances ranging from 288,840 to 46 million, and the number of features ranging from 6 to 64. Additionally, each dataset is naturally divided into multiple parts based on categorical attributes, and each part is further subdivided into training and test sets. For instance, the M5 dataset is split into 70 parts, each corresponding to different stores and departments. To construct scenarios with numerous learnwares, we develop 2,000 models for each dataset by repeating the following process. Specifically, for each dataset with C parts, we first randomly select an integer M from the range $[1, C/2]$, then randomly choose M parts from the C parts. Using the corresponding training data, we randomly select one of three model types, namely Random Forest [4], XGBoost [7], and LightGBM [14], and train the model by randomly selecting hyperparameters from a predefined set.

Evaluation of Learnware Identification. To investigate the impact of learnware admission management on identification performance, we construct 100 user tasks for each dataset by repeating the following procedure. For each dataset with C parts, we randomly select $M \in [1, C/2]$ parts in a manner similar to the above model generation process, and the user task consists of the corresponding test data for the selected parts. For each scenario with T user tasks, we evaluate the system's performance by computing the average loss across all tasks, i.e., $\sum_{i=1}^{T}\text{loss}_i/T$, where loss_i is the loss of identified models on the i-th user task. We use error rate and root-mean-square error (RMSE) as the loss functions for classification and regression tasks, respectively. It is important to note that the test instances of each user task are unseen by all previously developed learnwares.

Configurations. For the generation of RKME specifications, we set the specification size $n = 100$ and employ the Gaussian kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\gamma\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2)$ with $\gamma = 0.1$.

5.2 Evaluating the Effectiveness and Impact of Dynamic Learnware Filtering

Here, we evaluate our approach from multiple perspectives, including its impact on best identification performance, system size, and some learnware identification methods. To standardize these

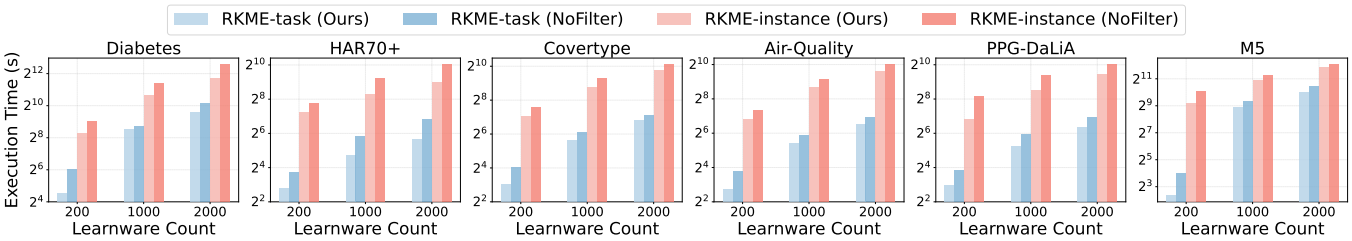

Figure 3: Comparison of identification efficiency between our approach and the baseline NoFilter for two representative learnware identification methods, with the number of uploaded learnwares ranging from 200 to 2,000 in each scenario.

Table 1: Ratio (%) of the two metrics (best identification performance and system size) of methods to those of the baseline NoFilter. As the learnware count increases up to 2,000, the results are presented as the mean and standard deviation. The best results of size reduction are emphasized in bold.

Scenario	CheckRKME / NoFilter		Ours / NoFilter	
	Best Perf.	System Size	Best Perf.	System Size
Diabetes	100.03 ± 0.04	95.60 ± 1.34	100.79 ± 0.12	**55.99** ± 5.90
HAR70+	103.19 ± 1.34	73.05 ± 9.48	100.64 ± 0.24	**47.27** ± 5.38
Covertype	100.04 ± 0.10	99.37 ± 0.56	100.30 ± 0.12	**72.09** ± 5.87
Air-Quality	100.33 ± 0.10	85.63 ± 5.09	100.35 ± 0.11	**68.99** ± 6.71
PPG-DaLiA	100.58 ± 0.37	95.29 ± 2.57	100.68 ± 0.12	**64.00** ± 1.68
M5	100.00 ± 0.01	99.11 ± 0.43	100.35 ± 0.37	**71.64** ± 13.81

Table 2: Ratio (%) of identification performance for two learnware identification methods (Ours / NoFilter). The results are presented for learnware counts ranging from 200 to 2,000.

Scenario	RKME-task			RKME-instance		
	200	1,000	2,000	200	1,000	2,000
Diabetes	99.98	101.67	101.09	100.48	102.65	101.04
HAR70+	101.05	104.79	103.90	100.02	100.41	104.32
Covertype	86.01	93.40	96.11	89.85	95.51	94.50
Air-Quality	97.76	100.34	100.52	96.77	98.57	100.31
PPG-DaLiA	98.86	102.07	102.14	99.16	97.12	97.95
M5	88.67	108.92	109.97	87.21	106.92	108.56
Mean	95.39	101.87	102.29	95.58	100.19	101.11

diverse metrics, we introduce the baseline method NoFilter, which admits all learnwares without any examination. For each metric, we then compute a relative ratio by dividing the metric's value by the corresponding baseline value. For example, the ratio of system size (Ours / NoFilter) indicates the system size of our approach compared to the baseline method.

Impact on Best Performance and System Size. For each user task, the best identification performance is defined as the minimum loss among all existing learnwares, representing the best possible performance of the system. Since the best identification performance typically improves as system size increases, we compare these two metrics side by side, with the number of uploaded learnwares continuously increasing up to 2,000. As illustrated in Figure 2, compared with the baseline NoFilter, our approach consistently and significantly reduces system size (an average size reduction of 27.91% to 52.73% across different scenarios) while maintaining nearly the same best identification performance (an average ratio of 100.30% to 100.79%). These results demonstrate the effectiveness of our approach in reducing system size while maintaining the best possible performance of the system.

Furthermore, a question may arise: Is the significant reduction in system size due to the presence of many learnwares with identical training tasks? To further validate our approach, we introduce a competitive method, CheckRKME, which identifies redundant learnwares whose RKME specifications are similar to those of existing learnwares. Table 1 presents the comparison results, indicating that our approach significantly outperforms CheckRKME in reducing system size, while both methods maintain similar best identification performance compared to NoFilter. Since CheckRKME can only eliminate learnwares trained on similar tasks, these results further demonstrate that our approach offers a precise characterization of model capabilities and can detect redundant learnwares even when their original tasks differ from those of existing ones.

Impact on Learnware Identification Methods. We further evaluate the impact of our approach on two representative learnware identification methods in terms of both identification performance and efficiency. Specifically, RKME-task [28] and RKME-instance [28], are designed to identify single and multiple learnwares, respectively, using RKME specifications. Table 2 presents the ratio of our approach to NoFilter in terms of the loss on user tasks for both identification methods, with the average ratio ranging from 95.39% to 102.29%. A ratio below 100% indicates improved identification performance, while a ratio above 100% suggests a slight decline. The results demonstrate that our approach does not significantly affect the efficacy of these representative methods and even improves their identification performance in many scenarios.

For learnware identification efficiency, we compare the total execution time of the two learnware identification methods using our approach and the baseline NoFilter in Figure 3. The results indicate that our approach significantly reduces the total execution time for both methods, with the average time reduction ranging from 32.34% to 52.95% for RKME-task and from 26.28% to 46.37% for RKME-instance across different scenarios. These findings demonstrate that our approach significantly improves the efficiency of learnware identification while maintaining or even enhancing the identification performance of these representative methods.

5.3 Practical System-Level Validation in Heterogeneous Feature Spaces

While the previous section demonstrated our dynamic filtering mechanism's effectiveness, it is crucial to validate its practical advantages for the learnware dock system, especially in more complex,

Table 3: Average RMSE loss and total average time (s) for solving a single user task on heterogeneous user tasks (10-2000 labeled samples). Mean ± Std. Dev. (over 3 repetitions). Beimingwu* denotes the method with our proposed dynamic filtering.

Method	10		50		100		200		500		1000		2000	
	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)	RMSE	Time (s)
LightGBM	4.60	**0.07**	4.18	**0.15**	3.94	**0.50**	3.72	**0.54**	3.31	**1.13**	3.08	2.73	2.94	4.03
(Std Dev)	(0.52)	(0.01)	(0.61)	(0.02)	(0.57)	(0.33)	(0.58)	(0.18)	(0.46)	(0.35)	(0.41)	(1.57)	(0.35)	(0.77)
TabPFN v2	3.88	27.53	3.52	31.51	3.29	77.10	3.21	86.80	2.97	146.15	2.94	168.96	3.05	165.98
(Std Dev)	(0.47)	(0.04)	(0.53)	(3.17)	(0.44)	(5.82)	(0.60)	(10.16)	(0.42)	(15.31)	(0.46)	(27.77)	(0.47)	(3.57)
Beimingwu	**3.40**	1.39	**3.29**	1.42	**2.80**	1.21	**3.02**	1.24	**2.67**	1.37	**2.61**	1.58	**2.57**	1.31
(Std Dev)	(0.30)	(0.41)	(0.19)	(0.46)	(0.29)	(0.39)	(0.69)	(0.45)	(0.35)	(0.49)	(0.37)	(0.46)	(0.38)	(0.49)
Beimingwu*	**3.40**	1.22	3.30	1.11	2.84	1.13	**3.02**	1.14	2.69	1.30	**2.61**	**1.28**	**2.57**	**1.19**
(Std Dev)	(0.30)	(0.41)	(0.19)	(0.46)	(0.31)	(0.39)	(0.69)	(0.45)	(0.33)	(0.49)	(0.37)	(0.46)	(0.38)	(0.49)

real-world-like scenarios. Therefore, we adopt an established test scenario from the Beimingwu [25] system, which focuses on heterogeneous feature spaces. This scenario contains 265 learnwares spanning five distinct feature spaces, derived from two representative sales forecasting datasets: PFS [13] and Corporacion [12]. For evaluation, we use three different feature engineering methods for ten stores in the M5 [17] dataset, creating 30 user tasks, whose feature spaces differ from those of the learnwares. We then compare the performance of the learnware dock system (Beimingwu [25], incorporating our dynamic filtering mechanism) against strong non-learnware baselines, LightGBM [14] and TabPFN v2 [11].

System Slimming and Identification Efficiency. The key benefits enabled by our dynamic filtering approach are system slimming and identification efficiency. In this heterogeneous scenario, applying our filtering reduced the system size from 265 models to 166, achieving a significant 37.36% reduction in storage needs. This slimming, in turn, decreased the average single search time from 0.39 ± 0.08 seconds to 0.25 ± 0.02 seconds, a 35.75% improvement in search efficiency. These results underscore the effectiveness of our approach in managing diverse model systems by reducing redundancy and improving responsiveness.

Predictive Performance. We evaluate predictive performance using average RMSE loss across user tasks, varying user labeled data from 10 to 2,000 samples. The results, presented in Table 3, demonstrate that the learnware dock system (Beimingwu) consistently outperforms the LightGBM and TabPFN v2 baselines across limited labeled data scenarios. Crucially, our dynamic filtering method (indicated by Beimingwu*) successfully maintains this high level of system performance while achieving system slimming.

Runtime Efficiency. Practical deployment demands efficient task solving alongside good performance. We measure the total time taken (in seconds) by each method to address a single user task (results in Table 3). The learnware approach (Beimingwu*) exhibits remarkable time efficiency, significantly outperforming TabPFN v2, which has high prediction overhead on large user tasks (approx. 100k points each). While LightGBM's runtime notably increases with more training data, our method offers comparable, consistent efficiency and strong predictive performance.

These combined findings highlight that the learnware paradigm, supported by effective system management like our dynamic filtering, can offer practical advantages in performance, efficiency, and scalability, especially within limited labeled data scenarios.

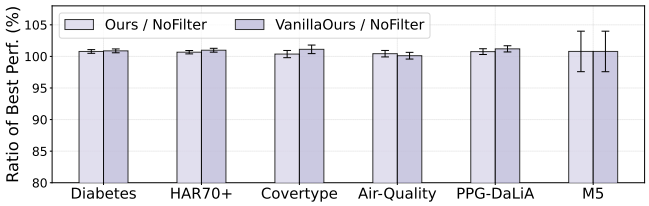

Figure 4: Ablation study on the impact of acceleration techniques on the best identification performance.

5.4 Ablation Study

To further investigate the impact of the efficient and scalable method proposed in Sections 4.1 and 4.2, we conduct an ablation study by comparing the performance and efficiency of our approach with and without these acceleration techniques. Specifically, we introduce VanillaOurs as a variant of our approach that detects redundant learnwares through a brute-force check of Criteria 4.2, without leveraging the structural organization of learnwares.

Impact on Performance and System Size. While our acceleration techniques enable efficient detection of redundant learnwares without traversing all existing ones, potentially introducing approximation errors, we assess their impact on the best identification performance and system size. As shown in Figure 4, the best identification performance of Ours is nearly identical to VanillaOurs, with both maintaining a ratio close to 100% relative to NoFilter. Meanwhile, Figure 5 reveals that the growth rate of system size in Ours aligns with or falls below VanillaOurs in most scenarios, exhibiting only marginal increases in limited cases while remaining significantly lower than NoFilter. These findings demonstrate that the impact on performance and system size induced by acceleration mechanisms remains constrained within our approach.

Effect of Acceleration on Efficiency. We further assess the efficiency of our approach, with and without acceleration techniques, in terms of cumulative runtime as the number of uploaded learnwares increases to 2,000. As shown in Figure 6, these acceleration techniques significantly improve the efficiency of our approach, achieving speedups ranging from 2.2× to 18.9× over VanillaOurs across different scenarios. These results validate the effectiveness of our acceleration techniques in enhancing the efficiency of learnware admission management while maintaining the performance of detecting redundant learnwares.

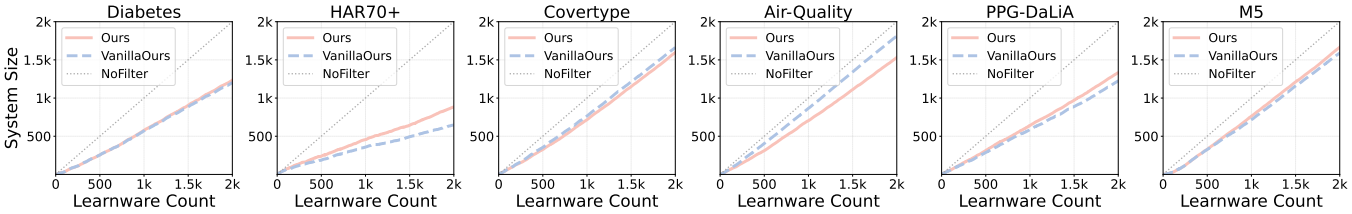

Figure 5: Ablation study on the impact of acceleration techniques on system size as the number of uploaded learnwares continuously increases up to 2,000 in each scenario. NoFilter is included as a baseline for comparison.

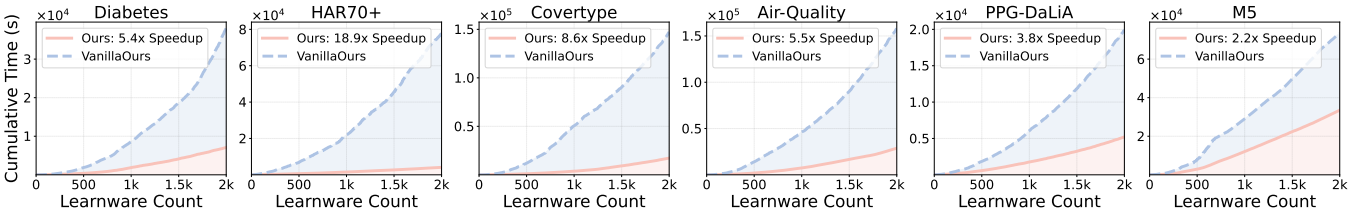

Figure 6: Ablation study of accelerated vs. non-accelerated methods: Runtime efficiency with learnware count up to 2,000.

6 Related Work

Recently, model pools and hubs have experienced substantial growth, with the Hugging Face platform notably hosting over a million models. As envisioned by the learnware paradigm, with the growing number of models from various tasks, effective model admission criteria, management, and identification become increasingly important and challenging. These model pools generally operate as Git-based remote hosting services, admitting models without restrictions on their capabilities and managing them similar to code repositories, with semantic descriptions capturing model information. Within this context of model selection from large model repositories, some works such as HuggingGPT [21] and ToolLLM [19] propose using popular large language models (LLMs) [5] to identify helpful models or tools based on their natural language descriptions on the platform, without characterizing model capabilities from statistical perspectives. Other studies assess the reusability or transferability of pre-trained models without fine-tuning [8, 30, 32]. Such approaches, which typically require running all candidate pre-trained models on user data, ignore data privacy and are impractical in real-world scenarios involving numerous models.

Note that the learnware paradigm [34] was proposed much earlier than the appearance of these model pools. Actually, these model pools can be regarded as a naive realization of a learnware dock system, where the learnware specification is realized as language descriptions. Furthermore, the learnware paradigm has designs for protecting the original training data of developers or users, and provides possibilities for assembling multiple models to serve users; particularly, some tasks that have not been considered by existing models can be addressed by assembling existing models in the learnware paradigm. More profoundly, the learnware paradigm utilizes statistical specifications, e.g., the RKME specification [35], to characterize the intrinsic capabilities of models. This enables a learnware dock system, such as the Beimingwu[1] platform [25], to effectively identify and reuse high-performing models for user tasks without accessing user raw data or exhaustively evaluating every candidate model, achieving active management of accommodated models rather than passive hosting. As these learnware dock systems expand, the effective model admission criteria become crucial for sustained system efficiency and scalability. This work specifically targets this issue, introducing a method for dynamic learnware filtering based on model capability coverage to achieve system slimming and improve learnware identification efficiency.

[1]https://github.com/Learnware-LAMDA

7 Conclusion

This paper presents the first attempt to establish learnware admission criteria and dynamically filter redundant learnwares from the perspective of model capability coverage. To achieve this, we structurally organize task information from all learnwares into an RKME cover tree, enabling the evaluation of model capabilities across a continuously expanding task set. Building on this structure and capability representation, we propose a method that is both efficient and scalable for filtering redundant learnwares dynamically, without requiring traversal of the entire system. The effectiveness and efficiency of our approach are validated through theoretical analysis, extensive experiments involving over ten thousand simulated models across various real-world scenarios, and an ablation study. Our approach lays a crucial foundation for the practical realization of large-scale, sustainable learnware dock systems, significantly enhancing their long-term viability and utility. Future research could extend this work by investigating adaptive filtering thresholds.

Acknowledgments

This research was supported by National Natural Science Foundation of China (62250069) and the Collaborative Innovation Center of Novel Software Technology and Industrialization. The authors would like to thank Peng Tan, Jia-Wei Shan, and Hai-Tian Liu for helpful discussions. We are also grateful to the anonymous reviewers for their helpful comments.

References

[1] Francis R. Bach, Simon Lacoste-Julien, and Guillaume Obozinski. 2012. On the equivalence between herding and conditional gradient algorithms. In *Proceedings of the 29th International Conference on Machine Learning*. 1355–1362.

[2] Alina Beygelzimer, Sham M. Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*. 97–104.

[3] Jock A Blackard and Denis J Dean. 1999. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture* 24, 3 (1999), 131–151.

[4] Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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.

[6] Nilka Rios Burrows, Israel Hora, Linda S. Geiss, Edward W. Gregg, and Ann Albright. 2017. Incidence of end-stage renal disease attributed to diabetes among persons with diagnosed diabetes—United States and Puerto Rico, 2000–2014. *MMWR. Morbidity and Mortality Weekly Report* 66 (2017), 1165–1170.

[7] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[8] 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.

[9] Yury Elkin and Vitaliy Kurlin. 2023. A new near-linear time algorithm for k-nearest neighbor search using a compressed cover tree. In *Proceedings of the 40th International Conference on Machine Learning*. 9267–9311.

[10] 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.

[11] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. 2025. Accurate predictions on small data with a tabular foundation model. *Nature* 637, 8044 (2025), 319–326.

[12] Kaggle. 2017. Corporación favorita grocery sales forecasting. https://www.kaggle.com/c/favorita-grocery-sales-forecasting. Accessed: 2025-04-15.

[13] Kaggle. 2018. Predict future sales. https://kaggle.com/competitions/competitive-data-science-predict-future-sales. Accessed: 2025-04-15.

[14] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*. 3146–3154.

[15] Hao-Yi Lei, Zhi-Hao Tan, and Zhi-Hua Zhou. 2024. On the ability of developers' training data preservation of learnware. In *Advances in Neural Information Processing Systems 37*. 36471–36513.

[16] 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.

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

[18] Krikamol Muandet, Kenji Fukumizu, Bharath K. Sriperumbudur, and Bernhard Schölkopf. 2017. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning* 10, 1-2 (2017), 1–141.

[19] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *Proceedings of the 12th International Conference on Learning Representations*.

[20] Attila Reiss, Ina Indlekofer, Philip Schmidt, and Kristof Van Laerhoven. 2019. Deep PPG: Large-scale heart rate estimation with convolutional neural networks. *Sensors* 19, 14 (2019), 3079.

[21] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems 36*. 38154–38180.

[22] Alexander J. Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. 2007. A Hilbert space embedding for distributions. In *Proceedings of the 18th International Conference on Algorithmic Learning Theory*. 13–31.

[23] Peng Tan, Hai-Tian Liu, Zhi-Hao Tan, and Zhi-Hua Zhou. 2024. Handling learnwares from heterogeneous feature spaces with explicit label exploitation. In *Advances in Neural Information Processing Systems 37*. 12767–12795.

[24] 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.

[25] 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*. 5773–5782.

[26] Zhi-Hao Tan, Zi-Chen Zhao, Hao-Yu Shi, Xin-Yu Zhang, Peng Tan, Yang Yu, and Zhi-Hua Zhou. 2025. Learnware of language models: Specialized small language models can do big. arXiv:2505.13425

[27] Astrid Ustad, Aleksej Logacjov, Stine Øverengen Trollebø, Pernille Thingstad, Beatrix Vereijken, Kerstin Bach, and Nina Skjæret-Maroni. 2023. Validation of an activity type recognition model classifying daily physical behavior in older adults: The HAR70+ model. *Sensors* 23, 5 (2023), 2368.

[28] 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.

[29] 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.

[30] Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. 2021. LogME: Practical assessment of pre-trained models for transfer learning. In *Proceedings of the 38th International Conference on Machine Learning*. 12133–12143.

[31] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. 2017. Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473, 2205 (2017), 20170457.

[32] 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*. 13692–13719.

[33] 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.

[34] Zhi-Hua Zhou. 2016. Learnware: on the future of machine learning. *Frontiers of Computer Science* 10, 4 (2016), 589–590.

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

Supplementary Materials

In the appendix, we first summarize the primary notations used in this work. Next, we provide the proofs of theoretical results and include the additional experimental details and results.

A Notations

The major notations used in this work are summarized in Table 4.

Table 4: Major notations used in this work.

Category	Notation	Description
Developer	$\mathcal{D}_i$	The original training distribution of the i-th developer over the input space $\mathcal{X}$.
	$D_i := (\boldsymbol{X}_i, \boldsymbol{y}_i)$	The local dataset of the i-th developer, sampled from the distribution $\mathcal{D}_i$.
	(f_i, R_i)	The i-th learnware, where f_i is the model and R_i is the RKME specification.
	m, n	The sizes of the training dataset D_i and RKME specification R_i, respectively.
Loss	$\mathcal{L}_{\mathcal{D}_i}(f, f')$	The expected loss between models f and f' on the distribution $\mathcal{D}_i$.
	$\widetilde{\mathcal{L}}_{R_i}(f, f')$	The empirical loss between models f and f' on the RKME specification R_i.
Admission Criteria	$\mathcal{S}$	The set of existing learnwares in the system, i.e., $\{(f_i, R_i)\}_{i=1}^N$.
	$\mathcal{I}_f(\mathcal{S})$	The task set that the model f excels at, containing R_i if $\widetilde{\mathcal{L}}_{R_i}(f, f_i) \leq \xi, (f_i, R_i) \in \mathcal{S}$.
RKME Cover Tree	$d_{\mathcal{H}_k}(R_i, R_j)$	The distance metric between any two RKME specifications, derived within the RKHS $\mathcal{H}_k$.
	$\mathcal{T}(\mathcal{R})$	The RKME cover tree built on the set of RKME specifications $\mathcal{R}$.
	$l : \mathcal{R} \mapsto \mathbb{Z}$	The level function that maps each RKME specification to its level in the cover tree.
	$\mathrm{Children}(p)$	The set of children nodes of the node p in the RKME cover tree $\mathcal{T}(\mathcal{R})$.
	$\mathrm{Next}(p, i)$	The largest level j such that $j < i$ and the node p has children at level j.
	$\mathrm{Covdist}(p)$	The maximum distance between the node p and its descendants in the RKME cover tree.
Learnware Filtering	$\mathcal{M}$ / $\mathcal{M}'$	The set of model f_i whose task set $\mathcal{I}_{f_i}(\mathcal{S})$ is a superset of that of the new model f_{new}.
	$\mathcal{U}_R(\mathcal{S})$	The set of model f_i that excels at the task indicated by R, i.e., $R \in \mathcal{I}_{f_i}(\mathcal{S})$.
	$\mathcal{V}$ / $\mathcal{V}'$	The set of model f_i whose task set $\mathcal{I}_{f_i}(\mathcal{S})$ satisfying $\mathcal{I}_{f_i}(\mathcal{S}) \subseteq \mathcal{I}_{f_{\text{new}}}(\mathcal{S})$.

B Proofs

B.1 Proof of Theorem 4.1

To prove Theorem 4.1, we first present several lemmas on the convergence rates of the empirical KME $\hat{\mu}_{\mathcal{D}}$ to the KME $\mu_{\mathcal{D}}$, and the RKME $\tilde{\mu}_{\mathcal{D}}$ to the empirical KME $\hat{\mu}_{\mathcal{D}}$, for any task distribution $\mathcal{D}$ over the input space $\mathcal{X}$, as defined in Section 2.

Lemma B.1 (Theorem 3.4 of Muandet et al. [18])**.** *Assuming that* $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) \leq B_{\mathcal{H}}$*, with probability at least* $1 - \delta$*, where* $\delta \in (0, 1)$*, the following holds:*

$$\|\mu_{\mathcal{D}} - \hat{\mu}_{\mathcal{D}}\|_{\mathcal{H}_k} \leq \sqrt{\frac{B_{\mathcal{H}}}{m}} + \sqrt{\frac{2B_{\mathcal{H}} \log \frac{1}{\delta}}{m}},$$

where m *denotes the local dataset size.*

Lemma B.1 shows that $\hat{\mu}_{\mathcal{D}}$ converges to $\mu_{\mathcal{D}}$ at a rate of $O(1/\sqrt{m})$. We then derive the convergence rate of $\tilde{\mu}_{\mathcal{D}}$ to $\hat{\mu}_{\mathcal{D}}$ through a constructive proof, as presented in Lemma B.2.

Lemma B.2. *Assume* $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) \leq B_{\mathcal{H}}$*. Then, we have:*

$$\|\hat{\mu}_{\mathcal{D}} - \tilde{\mu}_{\mathcal{D}}\|_{\mathcal{H}_k} \leq 2\sqrt{\frac{2B_{\mathcal{H}}}{n}},$$

where n *denotes the RKME specification size.*

Proof of Lemma B.2. To analyze the convergence rate of $\tilde{\mu}_{\mathcal{D}}$ to $\hat{\mu}_{\mathcal{D}}$, we apply the kernel herding method [1], which generates a weighted mimic dataset $\{(\beta_j^{\text{mic}}, \boldsymbol{z}_j^{\text{mic}})\}_{j=1}^n$, where $\boldsymbol{\beta}^{\text{mic}} \in \Delta^n$, to approximate $\hat{\mu}_{\mathcal{D}}$. According to Bach et al. [1], assuming $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) \leq B_{\mathcal{H}}$, we have:

$$\|\hat{\mu}_{\mathcal{D}} - \tilde{\mu}_{\mathcal{D}}^{\text{mic}}\|_{\mathcal{H}_k} \leq 2\sqrt{2B_{\mathcal{H}}/n},$$

where $\tilde{\mu}_{\mathcal{D}}^{\text{mic}} = \sum_{j=1}^n \beta_j^{\text{mic}} k(\boldsymbol{z}_j^{\text{mic}}, \cdot)$. Since the mimic dataset generated by the kernel herding algorithm is a suboptimal solution to the optimization problem for generating the RKME specification, as shown in Eq. (1), the convergence rate of the RKME $\tilde{\mu}_{\mathcal{D}}$ to the empirical KME $\hat{\mu}_{\mathcal{D}}$ can be bounded as follows:

$$\|\hat{\mu}_{\mathcal{D}} - \tilde{\mu}_{\mathcal{D}}\|_{\mathcal{H}_k} \leq \|\hat{\mu}_{\mathcal{D}} - \tilde{\mu}_{\mathcal{D}}^{\text{mic}}\|_{\mathcal{H}_k} \leq 2\sqrt{2B_{\mathcal{H}}/n},$$

which completes the proof. □

Lemma B.2 presents the convergence rate of the RKME $\tilde{\mu}_{\mathcal{D}}$ to the empirical KME $\hat{\mu}_{\mathcal{D}}$ as $O(1/\sqrt{n})$. Additionally, when the RKHS $\mathcal{H}_k$ is finite-dimensional, the convergence rate improves to $O(e^{-n})$ [1, 33]. Based on these convergence rates, we now proceed to prove Theorem 4.1, where the subscript $\mathcal{H}_k$ is omitted in $\langle\cdot,\cdot\rangle_{\mathcal{H}_k}$ and $\|\cdot\|_{\mathcal{H}_k}$ when it is clear from the context.

Proof of Theorem 4.1. For all $f \in \mathcal{F}$, we have:

$$\mathcal{L}_{\mathcal{D}_i}(f, h) = \left\langle \mu_{\mathcal{D}_i}, \ell_{f,h} \right\rangle \leq \left\langle \mu_{\mathcal{D}_i}, \ell_{f,f_i} + \ell_{f_i,h} \right\rangle \leq \varepsilon + \left\langle \mu_{\mathcal{D}_i}, \ell_{f,f_i} \right\rangle,$$

where the first inequality holds due to the assumption that ℓ obeys the triangle inequality, and the second inequality is due to the fact that $\mathcal{L}_{\mathcal{D}_i}(f_i, h) = \left\langle \mu_{\mathcal{D}_i}, \ell_{f_i,h} \right\rangle \leq \varepsilon$ in Eq. (2). From this, we have:

$$\begin{aligned}
\mathcal{L}_{\mathcal{D}_i}(f, h) &\leq \varepsilon + \left\langle \mu_{\mathcal{D}_i} - \hat{\mu}_{\mathcal{D}_i} + \hat{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_i} + \tilde{\mu}_{\mathcal{D}_i}, \ell_{f,f_i} \right\rangle \\
&= \varepsilon + \left\langle \mu_{\mathcal{D}_i} - \hat{\mu}_{\mathcal{D}_i} + \hat{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_i}, \ell_{f,f_i} \right\rangle + \widetilde{\mathcal{L}}_{R_i}(f, f_i) \\
&\leq \varepsilon + \left\|\ell_{f,f_i}\right\| \left(\left\|\mu_{\mathcal{D}_i} - \hat{\mu}_{\mathcal{D}_i}\right\| + \left\|\hat{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_i}\right\|\right) + \widetilde{\mathcal{L}}_{R_i}(f, f_i) \\
&\leq \varepsilon + \widetilde{\mathcal{L}}_{R_i}(f, f_i) + O(1/\sqrt{m} + 1/\sqrt{n}), \qquad (11)
\end{aligned}$$

where Eq. (11) holds by Lemmas B.1 and B.2 and the assumption that ℓ_{f,f_i} is bounded. Then, we prove the other side:

$$\begin{aligned}
\widetilde{\mathcal{L}}_{R_i}(f, f_i) &= \left\langle \tilde{\mu}_{\mathcal{D}_i} - \hat{\mu}_{\mathcal{D}_i} + \hat{\mu}_{\mathcal{D}_i} - \mu_{\mathcal{D}_i} + \mu_{\mathcal{D}_i}, \ell_{f,f_i} \right\rangle \\
&\leq \underbrace{\left\|\ell_{f,f_i}\right\| \left(\left\|\tilde{\mu}_{\mathcal{D}_i} - \hat{\mu}_{\mathcal{D}_i}\right\| + \left\|\hat{\mu}_{\mathcal{D}_i} - \mu_{\mathcal{D}_i}\right\|\right)}_{(A_1)} + \underbrace{\left\langle \mu_{\mathcal{D}_i}, \ell_{f,f_i} \right\rangle}_{(A_2)}.
\end{aligned}$$

Since $(A_1) = O(1/\sqrt{m} + 1/\sqrt{n})$ holds in the same manner as Eq. (11), we derive (A_2) as follows:

$$(A_2) \leq \left\langle \mu_{\mathcal{D}_i}, \ell_{f,h} + \ell_{h,f_i} \right\rangle \leq \mathcal{L}_{\mathcal{D}_i}(f, h) + \varepsilon,$$

where the first inequality holds by the triangle inequality of ℓ, and the second inequality is due to Eq. (2). Thus, we obtain:

$$\left|\mathcal{L}_{\mathcal{D}_i}(f,h) - \widetilde{\mathcal{L}}_{R_i}(f,f_i)\right| = O\left(\varepsilon + 1/\sqrt{m} + 1/\sqrt{n}\right),$$

which completes the proof. □

B.2 Proof of Proposition 4.3

For simplicity, we assume the training datasets D_i and D_j have the same size m, and the RKME specifications R_i and R_j have the same size n. The proof of Proposition 4.3 is then presented as follows, where the subscript $\mathcal{H}_k$ is omitted in $\langle\cdot,\cdot\rangle_{\mathcal{H}_k}$ and $\|\cdot\|_{\mathcal{H}_k}$ when it is clear from the context.

Proof. With probability at least $1-\delta$, where $\delta \in (0,1)$, for all $f \in \mathcal{F}$, we have:

$$\begin{aligned}
&\left|\widetilde{\mathcal{L}}_{R_i}(f,f_i) - \widetilde{\mathcal{L}}_{R_j}(f,f_j)\right| \\
\le &\left|\widetilde{\mathcal{L}}_{R_i}(f,f_i) - \mathcal{L}_{\mathcal{D}_i}(f,h)\right| + \left|\mathcal{L}_{\mathcal{D}_i}(f,h) - \widetilde{\mathcal{L}}_{R_j}(f,f_j)\right| \\
\le &\left|\mathcal{L}_{\mathcal{D}_i}(f,h) - \mathcal{L}_{\mathcal{D}_j}(f,h) + \mathcal{L}_{\mathcal{D}_j}(f,h) - \widetilde{\mathcal{L}}_{R_j}(f,f_j)\right| + O(\eta) \\
\le &\left|\mathcal{L}_{\mathcal{D}_i}(f,h) - \mathcal{L}_{\mathcal{D}_j}(f,h)\right| + O(\eta), \qquad (12)
\end{aligned}$$

where these inequalities hold by Theorem 4.1. Then we derive:

$$\begin{aligned}
\left|\mathcal{L}_{\mathcal{D}_i}(f,h) - \mathcal{L}_{\mathcal{D}_j}(f,h)\right| &= \left|\left\langle \mu_{\mathcal{D}_i} - \mu_{\mathcal{D}_j}, \ell_{f,h}\right\rangle\right| \\
&\le \left\|\ell_{f,h}\right\| \cdot \left\|\mu_{\mathcal{D}_i} - \mu_{\mathcal{D}_j}\right\|. \qquad (13)
\end{aligned}$$

Since $\ell_{f,h}$ is bounded, we derive $\|\mu_{\mathcal{D}_i} - \mu_{\mathcal{D}_j}\|$ as follows:

$$\begin{aligned}
\left\|\mu_{\mathcal{D}_i} - \mu_{\mathcal{D}_j}\right\| &= \left\|(\mu_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_i}) + (\tilde{\mu}_{\mathcal{D}_j} - \mu_{\mathcal{D}_j}) + (\tilde{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_j})\right\| \\
&\le \left\|\mu_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_i}\right\| + \left\|\tilde{\mu}_{\mathcal{D}_j} - \mu_{\mathcal{D}_j}\right\| + \left\|\tilde{\mu}_{\mathcal{D}_i} - \tilde{\mu}_{\mathcal{D}_j}\right\| \\
&\le d_{\mathcal{H}_k}(R_i,R_j) + O(1/\sqrt{m} + 1/\sqrt{n}), \qquad (14)
\end{aligned}$$

where Eq. (14) holds by Lemmas B.1 and B.2. By combining Eq. (12), Eq. (13), and Eq. (14), we obtain:

$$\left|\widetilde{\mathcal{L}}_{R_i}(f,f_i) - \widetilde{\mathcal{L}}_{R_j}(f,f_j)\right| = O(d_{\mathcal{H}_k}(R_i,R_j) + \eta),$$

which completes the proof. □

C Additional Experimental Details and Results

This section offers an in-depth look at our experimental setup and presents supplementary results, further substantiating the findings from Section 5 and supporting our method's effectiveness.

C.1 Omitted Details on Settings

The predefined hyperparameter sets for each model type mentioned in Section 5.1 are presented as follows:

- Random Forest [4]: The hyperparameters including `max_depth`, `min_samples_split`, and `min_samples_leaf` are randomly selected from $\{(15,2,1),(8,5,2),(7,4,2),(10,10,5),(12,8,4)\}$.
- XGBoost [7]: The hyperparameters including `learning_rate`, `max_depth`, `subsample`, and `colsample_bytree` are randomly selected from $\{(0.01,6,0.8,0.8),(0.05,8,0.9,0.7),(0.1,10,0.7,0.9),(0.02,12,0.85,0.75),(0.03,15,0.6,0.6)\}$.
- LightGBM [14]: The hyperparameters including `num_leaves`, `max_depth`, and `learning_rate` are randomly selected from the predefined set $\{(31,5,0.01),(50,10,0.05),(64,15,0.1),(40,12,0.02),(80,20,0.03)\}$.

For any hyperparameters not explicitly mentioned above, we utilized the default values provided by their respective libraries. These varying hyperparameters ensure that the models are diverse and span a wide range of complexities. Note that the test instances for each user task remain unseen during model training.

Table 5: Relative search time (%) of methods vs. NoFilter using identification method RKME-task and RKME-instance.

Scenario	RKME-task		RKME-instance	
	CheckRKME	Ours	CheckRKME	Ours
Diabetes	93.67	**61.61**	92.45	**69.71**
HAR70+	62.04	**44.10**	74.66	**63.74**
Covertype	98.66	**79.90**	80.71	**72.90**
Air-Quality	76.82	**74.66**	78.01	**75.43**
PPG-DaLiA	93.16	**67.04**	89.57	**68.35**
M5	98.58	**83.06**	95.46	**85.28**
Mean	87.16	**68.40**	85.14	**72.57**

Table 6: Ratio (%) of two metrics of our method to those of NoFilter with varying performance threshold ξ.

(a) HAR70+ classification scenario.

Metric	$\xi = 0.02$	$\xi = 0.03$	$\xi = 0.04$	$\xi = 0.05$	$\xi = 0.06$
Best Perf.	100.26	100.64	101.10	101.15	101.62
System Size	55.23	47.27	41.07	36.52	33.70

(b) PPG-DaLiA regression scenario.

Metric	$\xi = 8.0$	$\xi = 9.0$	$\xi = 10.0$	$\xi = 11.0$	$\xi = 12.0$
Best Perf.	100.76	100.94	100.68	101.15	101.28
System Size	67.87	65.73	64.00	61.61	60.55

C.2 Detailed Runtime Efficiency Comparison

To further illustrate the efficiency advantage of our method, this section focuses on scenarios with 2,000 uploaded learnwares. At this scale, as noted in the main text, identification performance is consistent across methods, making search time a key metric for efficiency evaluation. Table 4 presents the relative search time (as a percentage) for CheckRKME and our proposed method compared to the NoFilter baseline. The data clearly shows that our approach significantly reduces the required search time, thus validating its effectiveness in enhancing learnware identification efficiency.

C.3 Analysis of the Parameter Stability

We conduct parameter stability experiments for the performance threshold ξ in both classification and regression scenarios. As shown in Table 6, a larger threshold leads to more pronounced system slimming effects, while the best performance remains relatively stable, demonstrating the stability of and the robustness of our method. Other hyperparameters, including the RKME specification size and kernel parameters, are fixed and reported in Section 5.1, consistent with related works [16, 23, 25].