Identifying and Reusing Learnwares Across Different Label Spaces

Jian-Dong Liu[1,2], **Zhi-Hao Tan**[1,2] and **Zhi-Hua Zhou**[1,2]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
{liujd, tanzh, zhouzh}@lamda.nju.edu.cn

Abstract

The *learnware paradigm* focuses on leveraging numerous established high-performing models to solve machine learning tasks instead of starting from scratch. As the key concept of this paradigm, a learnware consists of a well-trained model of any structure and a specification that characterizes the model's capabilities, allowing it to be identified and reused for future tasks. Given the existence of numerous real-world models trained on diverse label spaces, effectively identifying and combining these models to address tasks involving previously unseen label spaces represents a critical challenge in this paradigm. In this paper, we make the first attempt to identify and reuse effective learnware combinations for tackling learning tasks across different label spaces, extending their applicability beyond the original purposes of individual learnwares. To this end, we introduce a statistical class-wise specification for establishing similarity relations between various label spaces. Leveraging these relations, we model the utility of a learnware combination as a minimum-cost maximum-flow problem, and further develop fine-grained learnware identification and assembly methods. Extensive experiments with thousands of heterogeneous models validate our approach, demonstrating that reusing identified learnware combinations can outperform both training from scratch and fine-tuning a generic pre-trained model.

1 Introduction

Machine learning has achieved significant success in various practical fields, including medicine, robotics, and ecology. However, in classic machine learning paradigm, training a well-performing model from scratch still requires several challenging conditions, such as sufficient labeled data, adequate computational resources, and proficient training skills. Additionally, privacy and proprietary concerns obstruct data sharing among developers, limiting the potential of big models in many data-sensitive scenarios.

To tackle these issues simultaneously, *learnware* [Zhou, 2016; Zhou and Tan, 2024] was proposed for solving machine learning tasks in a novel paradigm. In this context, a learnware consists of a well-performing model of any structure and a *specification* which captures the model's specialty and utility in a certain representation, such as its statistical properties. Developers worldwide can submit their models trained on various tasks into a *learnware dock system* spontaneously, and the system helps generate specifications for each model to form learnwares. When facing a new user task, the system can identify and assemble helpful learnwares from all existing learnwares based on their specifications. The user can then reuse these learnwares with her own data to address her task instead of starting from scratch. It is important to note that the learnware dock system should not access the raw data of either model developers or users.

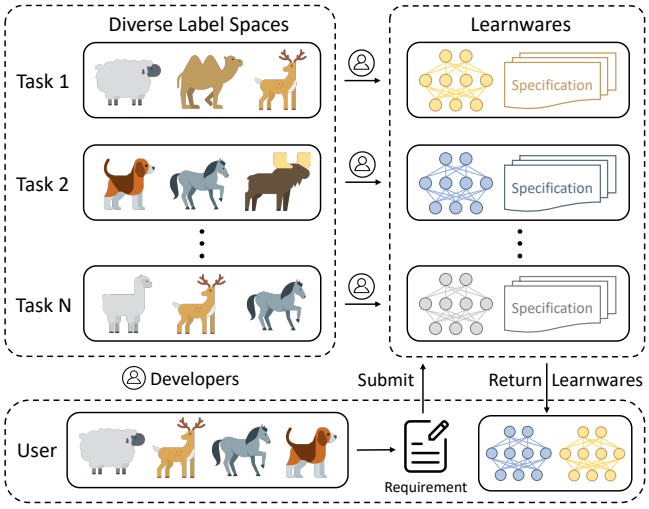

Figure 1: Given the existence of numerous learnwares from diverse label spaces, a critical challenge is to effectively identify and reuse helpful learnware combinations to address user tasks involving previously unseen label spaces.

To realize the vision depicted above, the critical challenge of learnware paradigm is to effectively identify and assemble useful learnwares within a learnware dock system for new user tasks, without accessing raw data. The key to the solution is *specification*, the core of this paradigm. Recently, the reduced kernel mean embedding (RKME) specification [Zhou and Tan, 2024] was proposed, characteriz-

ing model capabilities via distribution information. Based on this specification, several learnware search and reuse algorithms have been developed, and the efficacy of specification-based model selection and combination has been empirically and theoretically verified [Wu *et al.*, 2023; Liu *et al.*, 2024; Tan *et al.*, 2024a]. However, considering that numerous existing models and future user tasks come from diverse label spaces, an important problem in the learnware paradigm is still unsolved: *how to identify and reuse helpful learnware combinations for user tasks across different label spaces*, which greatly limits the scope of the learnware dock system. To illustrate, consider a scenario shown in Figure 1, where a user needs to classify classes such as sheep, deer, horses, and dogs. Although the system does not possess a single learnware trained specifically for classifying these four classes, there exists a learnware combination whose label spaces collectively encompass the user's required label space. If these relevant learnwares can be identified and combined to address the user task, it would significantly expand the scope of user tasks that can be handled, enabling the reuse of the combination beyond the limits of individual learnwares. With the initial learnware dock system recently built [Tan *et al.*, 2024b], tackling this problem has become more crucial.

In practice, the fundamental challenges of this problem mainly stem from two aspects: the unknown correspondence between different label spaces of learnwares and the user task, and the inability of the system to access raw data from model developers or users. Although accurate semantic information about each label space would make the problem easier, obtaining such information is particularly difficult due to the inherent ambiguity of natural language, the enormous number of learnwares in the system, and the additional complexity imposed on users in describing their requirements.

In this paper, to overcome these challenges without leveraging semantic information, our key insight is to represent model capabilities with statistical class-wise specifications and model the utility of a learnware combination to a user task from a perspective of class matching, thereby establishing relations between label spaces of each learnware and user tasks. Based on these relations, the system can handle user tasks involving previously unseen label spaces by identifying and assembling effective learnware combinations without access to raw user data. These learnwares collectively cover the required label space, providing an effective solution. The main contributions are summarized as follows:

- We make the first attempt to identify helpful learnware combinations across different label spaces to solve user tasks involving previously unseen label spaces, without accessing raw data from model developers or users. This enables the reuse of these combinations beyond the original purposes of individual learnwares, significantly expanding the scope of user tasks that can be handled.
- To represent a model's capabilities on each class, we extend the RKME specification into a class-wise version, capturing the conditional distribution and measuring similarity relations between different label spaces. Leveraging these relations, we model learnware combination utility for user tasks as minimum-cost maximum-flow problems, and further develop practical methods for identifying and reusing useful learnware combinations.
- Extensive experiments involving thousands of heterogeneous models validate the efficacy of our approach. Empirical results also show that reusing identified learnware combinations can outperform both training from scratch and fine-tuning a generic pre-trained model.

2 Problem Setup

The learnware paradigm consists of two stages: the submitting and deploying stages.

Submitting Stage. In this stage, N developers submit their models $\{f_i\}_{i=1}^N$ to the learnware dock system. Each model f_i is trained on a dataset $D_i = (\boldsymbol{X}_i, \boldsymbol{y}_i)$ over $\mathcal{X} \times \mathcal{Y}_i$, where $\mathcal{Y}_i$ denotes the label space unique to model f_i. In settings with heterogeneous label spaces, each model's label space $\mathcal{Y}_i$ can be distinct. Additionally, along with the well-trained model f_i, each developer also provides a specification S_i to the system, which is generated with assistance from the system.

Deploying Stage. In this stage, the user possesses an unlabeled dataset $D_u = \boldsymbol{X}_u$ defined over $\mathcal{X}$, and the label space associated with her task is $\mathcal{Y}_u$, which differs from that of all learnwares, i.e., $\mathcal{Y}_u \notin \{\mathcal{Y}_i\}_{i=1}^N$. Since there are various label spaces in the system, the user also has limited m labeled instances for each class in $\mathcal{Y}_u$. To tackle D_u, the user submits task requirements R_u to the system, which then returns a helpful learnware combination $\{f_i \mid i \in I\}$ with the number constraint $|I| \leq M$ based on specifications $\{S_i\}_{i=1}^N$. Subsequently, the user can solve her task by reusing these learnwares. The constraint about $|I|$ is natural and practical since the computing resources and time required for reusing learnwares are directly proportional to the number of learnwares.

Note that the learnware dock system cannot access raw data of either model developers or users. Besides, due to the decoupling of heterogeneous features and labels, identifying heterogeneous models typically requires finding a unified feature space [Tan *et al.*, 2023; Tan *et al.*, 2024a] before addressing diverse label spaces. Thus, this study assumes all models operate in a unified feature space $\mathcal{X}$ and can naturally expand to heterogeneous feature and label spaces in the future.

3 Our Approach

In this section, we first introduce the statistical class-wise specification, which characterizes model capabilities on different classes. Building on this, we then detail our comprehensive solution, focusing on algorithms for learnware identification and reuse.

3.1 Characterizing Model Capabilities

To effectively identify and assemble suitable model combinations across heterogeneous label spaces, the first step is to precisely characterize model capabilities on different classes. However, the lack of access to raw data from model developers or users poses a significant challenge.

The cornerstone of our solution is *specification*. While the RKME specification [Zhou and Tan, 2024] has shown efficacy in several learnware studies [Liu *et al.*, 2024; Tan *et al.*,

2024a] by capturing the entire data distribution, the recent implementation [Wu *et al.*, 2023] falls short in modeling the conditional data distribution for each class, restricting its efficacy in heterogeneous label space settings. To address this, we extend RKME into a class-wise version, the *class-wise RKME specification*, incorporating conditional distribution information and enabling accurate characterization of model capabilities on each class without exposing raw data.

Let $\Delta^n = \{\boldsymbol{\beta} \in \mathbb{R}^n \mid \mathbf{1}^\top \boldsymbol{\beta} = 1, \boldsymbol{\beta} \geq 0\}$ denote the n-dimensional simplex. For each class $c \in \mathcal{Y}$, let $\boldsymbol{X}_c = [\boldsymbol{x}_{c,1}; \boldsymbol{x}_{c,2}; \ldots; \boldsymbol{x}_{c,m_c}]$ represent the subset of dataset D belonging to class c and correctly classified by model f. Then the *class-wise RKME specification* $S = \{(\boldsymbol{\beta}_c \in \Delta^n, \boldsymbol{Z}_c \in \mathcal{X}^n)\}_{c\in\mathcal{Y}}$ is generated by solving:

$$\min_{\{\boldsymbol{\beta}_c, \boldsymbol{Z}_c\}_{c\in\mathcal{Y}}} \sum\nolimits_{c\in\mathcal{Y}} \left\| \hat{\mu}_c - \sum\nolimits_{j=1}^{n} \beta_{c,j} k(\boldsymbol{z}_{c,j}, \cdot) \right\|_{\mathcal{H}_k}^2, \quad (1)$$

where $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a kernel function with reproducing kernel Hilbert space $\mathcal{H}_k$, and $\hat{\mu}_c = \frac{1}{m_c} \sum_{i=1}^{m_c} k(\boldsymbol{x}_{c,i}, \cdot)$ is the empirical KME [Smola *et al.*, 2007] of class c. The reduced set size n, much smaller than the size of original dataset D, is empirically set to 10 in experiments. Inspired by [Wu *et al.*, 2023], we solve Eq. (1) using alternating optimization: $\boldsymbol{\beta}_c$ is updated via quadratic programming, while $\boldsymbol{Z}_c$ is optimized using gradient descent, as detailed in Algorithm 1.

Since the specification S is an extension of RKME across different classes, it retains the properties of RKME, such as protecting original data and providing robust defense against common inference attacks [Lei *et al.*, 2024]. Furthermore, it incorporates conditional distribution information, as demonstrated in the following proposition with proof in Appendix C.

Proposition 1. *Assume* $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) < \infty$. *Let* $\tilde{\mu}_c = \sum_{j=1}^{n} \beta_{c,j} k(\boldsymbol{z}_{c,j}, \cdot)$. *Then,* $\forall c \in \mathcal{Y}$, *we have* $\|\hat{\mu}_c - \tilde{\mu}_c\|_{\mathcal{H}_k} = \mathcal{O}(n^{-1/2})$, *where* n *is the reduced set size of specification* S.

As the empirical KME $\hat{\mu}_c$ captures the conditional distribution information of data correctly classified as class c by model f [Smola *et al.*, 2007], Proposition 1 indicates that the specification S accurately characterizes model capabilities on different classes, enabling precise learnware identification. In the submitting stage, each developer generates the *class-wise RKME specification* S_i with system assistance and submits the learnware (f_i, S_i) without disclosing their raw data.

3.2 Identifying Helpful Learnware Combinations

In the deploying stage, the user submits task requirements R_u to the system, expecting to receive helpful learnwares $\{f_i\}_{i\in I}$ for her task D_u, without leaking raw data. To achieve this, we first delve into the design of user task requirements and then present a practical learnware identification method.

User Task Requirements. Similar to model developers, the user can locally generate the *class-wise RKME specification* S_u using her limited m labeled data per class in $\mathcal{Y}_u$. To better capture the task's statistical properties, we further enhance the requirements R_u by estimating class probabilities.

For the user task, let $\boldsymbol{X}_u \in \mathcal{X}^{m_u}$ be the unlabeled dataset and $\{\boldsymbol{X}_c \in \mathcal{X}^m\}_{c\in\mathcal{Y}_u}$ the labeled dataset. Utilizing kernel mean embedding techniques [Smola *et al.*, 2007], we can estimate class probabilities $\boldsymbol{w} \in \Delta^{|\mathcal{Y}_u|}$ by solving the following problem via standard quadratic programming tools:

$$\min_{\boldsymbol{w}\in\Delta^{|\mathcal{Y}_u|}} \left\| \hat{\mu} - \sum\nolimits_{c\in\mathcal{Y}_u} \frac{w_c}{m} \sum\nolimits_{j=1}^{m} k(\boldsymbol{x}_{c,j}, \cdot) \right\|_{\mathcal{H}_k}^2, \quad (2)$$

where $\hat{\mu} = \frac{1}{m_u} \sum_{i=1}^{m_u} k(\boldsymbol{x}_{u,i}, \cdot)$ is the empirical KME of $\boldsymbol{X}_u$, and k is the kernel function from Eq. (1). The specification S_u and the class probabilities $\boldsymbol{w}$ together form the user requirements $R_u = (S_u, \boldsymbol{w})$, which describe statistical properties of the user task without exposing raw data, laying the foundation for establishing the relations between the heterogeneous label spaces of learnwares and the user task.

Learnware Identification. After receiving the requirements, the system is required to identify a useful learnware combination for the user task. Let $\mathcal{U}(I, R_u)$ denote the utility of a set of learnwares $\{f_i\}_{i\in I}$ to the user requirements R_u, and the learnware identification process can be formulated as

$$\max_{I\subseteq[N], |I|\leq M} \mathcal{U}(I, R_u). \quad (3)$$

To quantify $\mathcal{U}(I, R_u)$, we first measure the class similarity via the class-wise specification. Let $S_i = \{(\boldsymbol{\beta}_{c_1} \in \Delta^n, \boldsymbol{Z}_{c_1} \in \mathcal{X}^n)\}_{c_1\in\mathcal{Y}_i}$ and $S_u = \{(\boldsymbol{\beta}_{c_2} \in \Delta^n, \boldsymbol{Z}_{c_2} \in \mathcal{X}^n)\}_{c_2\in\mathcal{Y}_u}$ denote the specifications for the i-th learnware and the user task.

Algorithm 1 Specification Generation

Input: Local dataset D with label space $\mathcal{Y}$, model f, kernel function k, specification size n, iteration T.
Output: The class-wise RKME specification S.

for $c \in \mathcal{Y}$ **do**
  Obtain the data $\boldsymbol{X}_c$ in D that is correctly classified by model f as class c. Initialize $\boldsymbol{Z}_c$ by running k-means clustering on $\boldsymbol{X}_c$ with the number of clusters set to n.
  for $t = 1$ **to** T **do**
    Update $\boldsymbol{\beta}_c^{(t)}$ by using standard quadratic programming tools to minimize Eq. (1).
    Update $\boldsymbol{Z}_c^{(t)}$ by optimizing Eq. (1) with the gradient descent method.
  end for
end for
$S \leftarrow \{(\boldsymbol{\beta}_c^{(T)}, \boldsymbol{Z}_c^{(T)})\}_{c\in\mathcal{Y}}$.

Algorithm 2 Multiple Learnware Identification

Input: Learnwares $\{(f_i, S_i)\}_{i=1}^N$, user's local dataset $\boldsymbol{X}_u$ and $\{\boldsymbol{X}_c\}_{c\in\mathcal{Y}_u}$, constants M, K, λ.
Output: Identified models $\{f_i\}_{i\in I}$.

Based on the local dataset, the user generates specification S_u by solving Eq. (1) and estimates class probabilities $\boldsymbol{w}$ by solving Eq. (2), then submits task requirements $R_u = (S_u, \boldsymbol{w})$ to the system.
Initialize $I \leftarrow \{\}$, $I_0 \leftarrow \{\}$ and $\mathcal{U}(I_0, R_u) \leftarrow 0$.
for $t = 1$ **to** M **do**
  Obtain I_t by solving Eq. (4) with the successive shortest path algorithm applied to the minimum-cost maximum-flow formulation of $\mathcal{U}(I_t, R_u)$.
  If $\mathcal{U}(I_t, R_u) > \mathcal{U}(I, R_u)$ then $I \leftarrow I_t$; else exit loop.
end for
The system returns the helpful models $\{f_i\}_{i\in I}$.

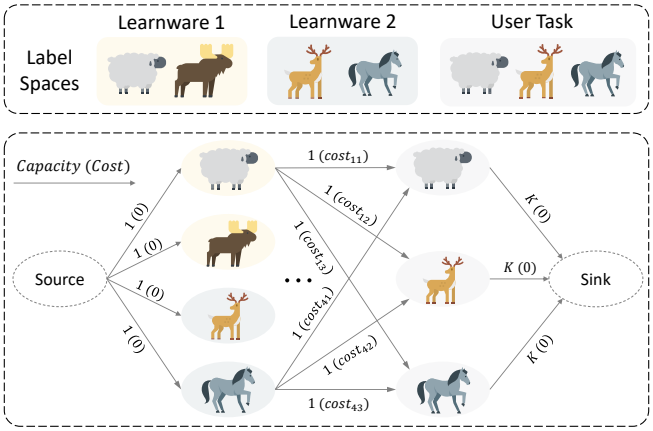

Figure 2: An illustration of modeling the utility of a learnware combination to the user task as a minimum-cost maximum-flow problem.

The similarity between class c_1 of the learnware and class c_2 of the user task, denoted as $\mathrm{sim}(S_{i,c_1}, S_{u,c_2})$, is defined as:

$$\lambda - \left\| \sum_{j=1}^{n} \beta_{c_1,j} k(\boldsymbol{z}_{c_1,j}, \cdot) - \sum_{j=1}^{n} \beta_{c_2,j} k(\boldsymbol{z}_{c_2,j}, \cdot) \right\|_{\mathcal{H}_k}^2,$$

where constant λ ensures non-negative similarity, and the latter term represents the Maximum Mean Discrepancy (MMD) between two RKMEs in RKHS $\mathcal{H}_k$. According to Proposition 1, this term quantifies the difference between the conditional distributions of the two classes.

Based on the similarity, we quantify $\mathcal{U}(I, R_u)$ from a perspective of class matching, aiming to identify learnwares whose combined label space fully covers the task label space. Specifically, we construct a bipartite graph with learnware classes on the left and user task classes on the right. The edge weight between class c_1 of learnware f_i and user class c_2 is $w_{c_2} \cdot \mathrm{sim}(S_{i,c_1}, S_{u,c_2})$, where w_{c_2} is the estimated probability of class c_2. Since each user class may have multiple similar learnware classes, we constrain that each user class can be matched by at most K learnware classes. Let variable $e^{(i)}_{c_1,c_2}$ indicate whether class c_1 of learnware f_i is matched to user class c_2. Then, the quantification of $\mathcal{U}(I, R_u)$ is modeled as the following optimization problem:

$$\begin{aligned}
\max \quad & \sum_{i\in\mathcal{I}, c_1\in\mathcal{Y}_i} \sum_{c_2\in\mathcal{Y}_u} e^{(i)}_{c_1,c_2} \cdot w_{c_2} \cdot \mathrm{sim}(S_{i,c_1}, S_{u,c_2}) \\
\text{s.t.} \quad & \sum_{c_2\in\mathcal{Y}_u} e^{(i)}_{c_1,c_2} \le 1, \quad \forall i \in \mathcal{I}, \forall c_1 \in \mathcal{Y}_i, \\
& \sum_{i\in\mathcal{I}, c_1\in\mathcal{Y}_i} e^{(i)}_{c_1,c_2} \le K, \quad \forall c_2 \in \mathcal{Y}_u, \\
& e^{(i)}_{c_1,c_2} \in \{0,1\}, \quad \forall i \in \mathcal{I}, \forall c_1 \in \mathcal{Y}_i, \forall c_2 \in \mathcal{Y}_u.
\end{aligned}$$

To solve this optimization problem, we transform it into an equivalent minimum-cost maximum-flow problem by treating the constraints as edge capacities and the negated edge weights as costs, as shown in Figure 2. The minimum cost can then be determined using the successive shortest path algorithm [Edmonds and Karp, 1972]. By taking the negative of this value, we obtain the maximum weight matching, which corresponds to the utility $\mathcal{U}(I, R_u)$. Further details of this modeling are provided in Appendix B.

Even with $\mathcal{U}(I, R_u)$ quantified, the problem in Eq. (3) remains NP-hard. To solve it practically, we employ a greedy algorithm to iteratively optimize Eq. (3). Specifically, at the t-th iteration, $I_t = I_{t-1} \cup \{i\}$ is obtained by solving:

$$\max_{i\in[N], i\notin I_{t-1}} \mathcal{U}\left(I_{t-1} \cup \{i\}, R_u\right). \tag{4}$$

The entire process of learnware identification is detailed in Algorithm 2, and its efficiency is analyzed in the following theorem, with the proof provided in Appendix C.

Theorem 1. *Assume the maximum size of the learnware label spaces* $\max_{i\in[N]} |\mathcal{Y}_i| \le \mathcal{C}_f$ *and the size of the user task label space* $|\mathcal{Y}_u| = \mathcal{C}_u$. *The time complexity of our learnware identification method is* $\mathcal{O}\left(N\mathcal{C}_f\mathcal{C}_u^2 \log\left(\mathcal{C}_f + \mathcal{C}_u\right)\right)$.

Since $\mathcal{C}_f$ and $\mathcal{C}_u$ are typically treated as constants, the time complexity of our method scales linearly with N, the dominant term. In contrast, the existing method [Wu *et al.*, 2023] for identifying multiple homogeneous learnwares has a time complexity of $\Omega(N^2)$, highlighting the efficiency of ours. Several methods can further accelerate our identification process. For instance, in the optimization of Eq. (4), traversing all indices $i \in [N]$ can be sped up via parallel computing. Additionally, techniques such as anchor learnwares [Zhou and Tan, 2024; Xie *et al.*, 2023] and specification index [Liu *et al.*, 2024] can substantially reduce the number of candidate learnwares N, further enhancing identification efficiency.

3.3 Reusing Learnware Combinations

While the identification phase seeks to provide learnwares whose combined label space covers the task label space, the reuse phase focuses on aligning and assembling these learnwares with the user task in a fine-grained manner. This assembly can be achieved by learning from the user's local labeled data, naturally excluding the need for learnware specifications during reuse and further protecting developer data privacy. Since effective reuse methods vary greatly with model structure, we design practical methods tailored for both non-deep and deep learning models, ensuring broad applicability.

Let $\{f_i : \mathcal{X} \mapsto \mathbb{R}^{|\mathcal{Y}_i|}\}_{i\in I}$ be the identified learnwares with diverse label spaces. For non-deep learning models (e.g., linear and tree-based), we employ methods inspired by stacking [Wolpert, 1992] and classifier adaptation [Li *et al.*, 2013]. Specifically, we augment the original features with logit vectors predicted by learnwares, which represent class probabilities. For instance, if $\boldsymbol{X}$ is the raw data and $I = \{1, 2\}$, the augmented data are $\boldsymbol{X}' = [\boldsymbol{X}, f_1(\boldsymbol{X}), f_2(\boldsymbol{X})]$, and a simple secondary classifier (e.g. logistic regression) is then trained on $\boldsymbol{X}'$. For deep learning models, each learnware is fine-tuned by freezing all layers except the last. Then an average ensemble method [Zhou, 2025] is applied, which averages the outputs of all learnwares for final predictions.

These proposed methods are computationally efficient with few learnable parameters. Moreover, they remain effective and robust even with limited labeled data, and improve continuously with more instances, as validated in Section 4.

4 Experiments

In this section, we develop thousands of models with diverse label spaces, spanning 22 real-world tabular and image sce-

Scenario	#Task Classes	#Models	Random	From-scratch	Linear-proxy	RKME-task	RKME-instance	Ours
CJS	6	20	99.23 ± 0.37	97.12 ± 1.20	99.32 ± 0.26	99.15 ± 0.64	99.54 ± 0.40	**99.64** ± 0.00
First-Order	6	20	81.43 ± 1.94	89.27 ± 2.88	80.47 ± 5.61	85.45 ± 1.33	96.41 ± 2.23	**97.96** ± 1.41
Covertype	7	35	53.45 ± 2.52	38.71 ± 2.81	66.71 ± 13.3	80.85 ± 3.62	90.45 ± 2.16	**98.79** ± 0.53
Fabert	7	35	39.78 ± 0.62	11.33 ± 0.00	41.73 ± 2.31	30.29 ± 1.22	61.03 ± 1.44	**65.48** ± 1.81
Steel-Fault	7	35	75.50 ± 5.11	83.32 ± 3.27	87.16 ± 2.92	84.50 ± 3.06	95.55 ± 0.92	**97.03** ± 1.30
MiceProtein	8	56	92.72 ± 2.04	84.26 ± 2.91	91.48 ± 1.93	92.50 ± 1.93	93.52 ± 0.83	**95.65** ± 1.19
Otto-Product	9	84	47.53 ± 3.17	47.71 ± 2.56	51.98 ± 5.37	54.71 ± 7.68	67.97 ± 5.58	**84.60** ± 2.83
Volkert	10	120	90.79 ± 2.17	80.00 ± 1.87	91.84 ± 1.35	89.95 ± 1.26	94.90 ± 0.78	**96.28** ± 1.16
CTG	10	120	61.22 ± 2.45	64.05 ± 2.74	67.16 ± 4.90	61.07 ± 3.13	71.81 ± 1.53	**82.51** ± 1.03
HAR	12	220	92.84 ± 1.50	91.29 ± 1.37	93.12 ± 0.79	93.45 ± 1.78	92.93 ± 2.38	**97.64** ± 0.28

Table 1: Average accuracy (%) on user tasks covering all classes in tabular scenarios. Each user task includes 10 labeled instances per class. The evaluations are repeated five times, and the results are presented as the mean and standard deviation. The best is emphasized in bold.

Scenario	#Task Classes	#Users	Random	From-scratch	Linear-proxy	RKME-task	RKME-instance	Ours
CJS	[4, 5]	21	99.42 ± 0.07	90.10 ± 1.76	99.34 ± 0.19	99.47 ± 0.05	99.56 ± 0.03	**99.66** ± 0.03
First-Order	[4, 5]	21	81.21 ± 2.62	56.26 ± 2.61	83.72 ± 2.60	88.94 ± 1.05	**99.55** ± 0.13	98.95 ± 0.33
Covertype	[5, 6]	28	63.00 ± 2.22	45.01 ± 1.68	78.19 ± 3.72	82.54 ± 3.28	**98.39** ± 0.40	98.10 ± 0.70
Fabert	[5, 6]	28	46.01 ± 0.42	17.64 ± 0.00	55.34 ± 1.72	51.85 ± 0.40	68.21 ± 0.95	**70.05** ± 1.12
Steel-Fault	[5, 6]	28	80.31 ± 1.09	85.65 ± 0.33	90.53 ± 1.24	90.51 ± 0.54	97.03 ± 0.39	**97.86** ± 0.25
MiceProtein	[6, 7]	36	93.61 ± 1.42	86.02 ± 1.24	93.73 ± 1.55	93.91 ± 1.27	97.29 ± 0.88	**97.40** ± 0.53
Otto-Product	[7, 8]	45	53.04 ± 0.98	48.94 ± 0.51	59.54 ± 1.20	65.24 ± 1.46	83.27 ± 0.96	**88.51** ± 0.62
Volkert	[8, 9]	55	90.80 ± 0.40	83.25 ± 0.62	92.40 ± 0.44	92.10 ± 0.22	95.58 ± 0.17	**96.56** ± 0.38
CTG	[8, 9]	55	64.08 ± 1.48	63.62 ± 1.15	69.80 ± 1.32	66.83 ± 1.76	77.47 ± 1.57	**84.59** ± 1.56
HAR	[10, 11]	78	93.04 ± 0.51	90.69 ± 0.77	94.96 ± 0.38	94.07 ± 0.58	94.07 ± 0.37	**97.67** ± 0.23

Table 2: Average accuracy (%) on user tasks covering partial classes in tabular scenarios, with the number of classes in user tasks varying within the range specified in the #Task Classes" column. Remaining experimental settings are consistent with Table 1.

narios. We compare with existing methods and conduct ablation studies to validate the efficacy of our approach.

4.1 Experimental Setup

Here we introduce some common experimental setups.

Evaluation. For a scenario with C users, methods are evaluated by the average classification accuracy $\sum_{i=1}^{C} \text{Acc}_i/C$, where Acc_i is the accuracy on the i-th user's unlabeled instances, which are unseen by all learnwares in the system.

Contenders. We compare our approach with five methods: two baselines, *Random* and *From-scratch*, and three related methods, *RKME-task* [Wu *et al.*, 2023], *RKME-instance* [Wu *et al.*, 2023], and *Linear-proxy* [Guo *et al.*, 2023]. *Random* randomly selects a learnware. *From-scratch* trains a new model from scratch with user labeled data and the same training algorithm as learnwares. *RKME-task* and *RKME-instance* identify single and multiple learnwares, respectively, using basic RKME specifications. *Linear-proxy* reduces the submitted model into a linear proxy model for identification. Since these contenders cannot simultaneously handle tabular and image scenarios across different label spaces, we enhance them with the reuse methods proposed in Section 3.3.

Configuration. We set the specification size n to 10 and use a Gaussian kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\gamma\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2)$ with $\gamma = 0.1$. For learnware search, K is chosen from $\{2, 3, 4\}$ for different datasets, with constants $M = 5$ and $\lambda = 100$.

4.2 Evaluation on Tabular Scenarios

Scenario Construction. For tabular scenarios, we develop 745 learnwares, each with a unique label space, derived from 10 real-world datasets on the OpenML [Vanschoren *et al.*, 2013] platform, spanning various domains such as healthcare, industrial, and biological fields. For each dataset, we select three classes from all classes, explore all possible combinations, and train models using LightGBM [Ke *et al.*, 2017] on the corresponding training data. For instance, for a dataset with 10 classes, we generate $\binom{10}{3} = 120$ models. This means most classes in user tasks are unseen by any single learnware.

User Task Covering All Classes. For each dataset, we test user tasks covering all classes, with each task D_u including the test set and 10 labeled data per class from the training set. Table 1 summarizes results from five repeats, showing that our method outperforms all contenders and significantly surpasses *From-scratch*, demonstrating the advantages of reusing identified learnwares with limited labeled data.

User Task Covering Partial Classes. We further test user tasks covering subsets of all classes. For a dataset with C classes, each task includes a subset of $C - 1$ or $C - 2$ classes from the test data. For example, the HAR dataset of 12 classes yields $\binom{12}{11} + \binom{12}{10} = 78$ tasks. Table 2 reports an 8/2 win/lose record, showing our method's efficacy even when the label spaces of user tasks and learnwares partially overlap.

User Task with Increasing Labeled Data. To further explore the benefits of our method, we test cases where labeled data per class increases from 10 to 5,000, or all available data if fewer than 5,000. Figure 3 shows that our method outperforms *From-scratch* with thousands of labeled data per class and even with all labeled data in some scenarios. This suggests that reusing identified learnwares could be more effective than training from scratch, even with sufficient data.

4.3 Evaluation on Image Scenarios

Scenario Construction. For image scenarios, we develop 300 heterogeneous learnwares from 12 real-world datasets: EuroSAT [Helber *et al.*, 2019], SVHN [Netzer *et al.*, 2011], CIFAR10/100 [Krizhevsky and Hinton, 2009], AID [Xia *et al.*, 2017], Pets [Parkhi *et al.*, 2012], Resisc45 [Cheng *et al.*, 2017], DTD [Cimpoi *et al.*, 2014], Food [Bossard *et al.*, 2014], Caltech101 [Li *et al.*, 2004], Flowers [Nilsback and

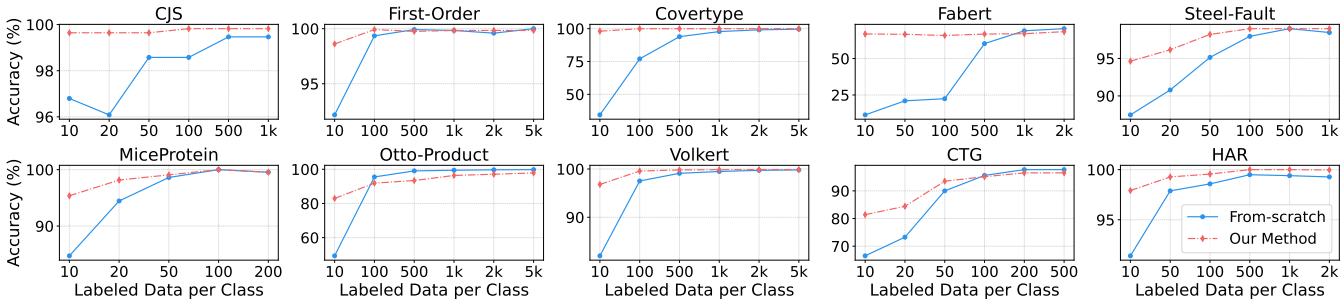

Figure 3: Average accuracy (%) on user tasks covering all classes in tabular scenarios, as the amount of labeled data per class increases.

Scenario	#Task Classes	Random	From-scratch	Linear-proxy	RKME-task	RKME-instance	ViT-ImageNet	Ours
SVHN	10	38.89 ± 0.64	12.72 ± 0.67	41.36 ± 9.88	41.54 ± 2.54	35.87 ± 10.8	20.15 ± 1.71	**78.19** ± 2.55
CIFAR10	10	65.54 ± 0.53	30.18 ± 2.91	75.92 ± 2.15	74.87 ± 0.73	82.61 ± 4.70	85.44 ± 1.16	**86.20** ± 2.64
EuroSAT	10	75.26 ± 1.74	46.54 ± 2.01	81.20 ± 1.74	80.77 ± 1.37	75.67 ± 4.57	78.50 ± 2.10	**87.86** ± 1.34
AID	30	76.83 ± 0.48	60.44 ± 1.53	81.52 ± 0.78	80.83 ± 0.96	85.13 ± 1.21	71.92 ± 1.30	**87.18** ± 0.90
Pets	37	86.41 ± 1.02	76.84 ± 2.28	87.23 ± 0.56	87.10 ± 0.81	88.34 ± 1.32	87.32 ± 0.37	**89.28** ± 0.75
Resisc45	45	68.62 ± 0.44	40.32 ± 1.77	76.32 ± 0.64	73.45 ± 0.75	77.06 ± 1.01	63.79 ± 1.14	**83.42** ± 0.95
DTD	47	55.46 ± 0.88	47.63 ± 1.54	59.03 ± 0.92	56.91 ± 1.13	61.37 ± 0.91	55.05 ± 0.63	**62.12** ± 1.02
CIFAR100	100	54.10 ± 0.25	20.25 ± 0.88	58.62 ± 0.83	57.93 ± 0.62	66.80 ± 1.50	63.65 ± 0.53	**68.16** ± 0.48
Food	101	53.21 ± 0.50	26.26 ± 0.64	59.00 ± 0.38	58.38 ± 0.29	67.32 ± 2.09	43.64 ± 0.85	**68.49** ± 0.80
Caltech101	101	84.91 ± 0.65	82.38 ± 0.60	86.45 ± 1.66	86.52 ± 0.94	88.62 ± 1.03	88.47 ± 0.85	**89.38** ± 0.99
Flowers	102	83.12 ± 0.53	69.61 ± 2.39	86.41 ± 0.39	85.36 ± 0.71	91.05 ± 1.86	81.56 ± 0.88	**93.83** ± 0.35
CUB2011	200	58.86 ± 0.24	47.48 ± 1.20	64.02 ± 0.27	61.95 ± 0.38	68.76 ± 0.29	55.31 ± 0.45	**68.98** ± 0.33

Table 3: Average accuracy (%) on user tasks covering all classes in image scenarios. Remaining details are consistent with Table 1.

Zisserman, 2008], and CUB2011 [Wah *et al.*, 2011]. These datasets span diverse domains, including plants, animals, and objects. For each dataset, we randomly select 20% to 50% of all classes and develop learnwares using corresponding training data. We perform 25 random selections per dataset, training ResNet50 [He *et al.*, 2016] models for 100 epochs using the SGD optimizer, with the initial learning rate chosen from $\{0.1, 0.01, 0.001\}$ and cosine annealing learning rate decay.

For specification generation, we use image features extracted by a generic pre-trained model, consistent with prior learnware studies [Wu *et al.*, 2023; Guo *et al.*, 2023]. Specifically, we apply a ViT-L/32 [Dosovitskiy *et al.*, 2021] model pre-trained on ImageNet [Russakovsky *et al.*, 2015]. For comparison, we introduce the *ViT-ImageNet* method, which fine-tunes the pre-trained model's last layer with user labeled data, aligning with our reuse strategy. We also compare with other fine-tuning methods, including full parameter tuning and LoRA [Hu *et al.*, 2022], detailed in Appendix D.

User Task Covering All Classes. Consistent with Section 4.2, we test user tasks covering all classes of the test set. Table 3 shows the superiority of our method across all scenarios, suggesting that identifying and reusing small proprietary models is more effective than training from scratch or fine-tuning a generic pre-trained model in few-shot settings.

User Task Covering Partial Classes. We further test tasks covering subsets of all classes. For each dataset, we randomly select 70% to 90% of all classes, with their test data as the user task. This process is repeated ten times to generate ten unique user tasks per dataset. Table 4 shows that our method consistently outperforms others, even when the user task label space completely differs from that of existing learnwares.

User Task with Increasing Labeled Data. Consistent with Section 4.2, we test cases with increasing labeled data per class. Figure 4 presents the superiority of our method in most scenarios, even when all labeled data are accessible.

4.4 Ablation Studies

Estimation of Class Probabilities. We compare our method using estimated probabilities $\boldsymbol{w}$ versus uniform probabilities on user tasks covering all classes. In class-balanced scenarios like CIFAR100 and Food, our method matches the accuracy of uniform probabilities. In class-imbalanced scenarios, it outperforms uniform probabilities by 6%, and in the most imbalanced scenario, Covertype, it leads by 45%. These results show the necessity of our class probability estimation.

Number of Identified Learnwares. We analyze how varying numbers of identified learnwares affect performance, as shown in Figure 5. Performance improves with more learnwares, but with diminishing returns. Reusing five learnwares generally yields satisfactory results, though cases like CTG, Otto-Product, and CIFAR100 could be further improved. In practice, the number of identified learnwares should depend on task complexity and available computational resources.

5 Related Work

Learnware. The learnware paradigm [Zhou, 2016; Zhou and Tan, 2024] proposes to build a large model platform comprising numerous high-performing models, enabling users to leverage existing models for their tasks. Each model is assigned a specification that characterizes its capabilities, allowing it to be identified for new tasks. Using the RKME specification, homogeneous learnwares can be identified by matching their data distributions with user tasks [Wu *et al.*, 2023]. This specification is proven to scarcely contain any original data and possesses robust defense against common

Scenario	#Task Classes	Random	From-scratch	Linear-proxy	RKME-task	RKME-instance	ViT-ImageNet	Ours
SVHN	[7, 9]	42.36 ± 0.71	13.93 ± 0.60	51.64 ± 6.46	54.19 ± 1.15	38.97 ± 5.59	23.41 ± 1.91	**73.59** ± 3.29
CIFAR10	[7, 9]	68.43 ± 0.58	36.16 ± 0.82	80.55 ± 1.04	81.61 ± 0.55	81.97 ± 1.59	87.01 ± 1.30	**87.41** ± 1.24
EuroSAT	[7, 9]	77.52 ± 1.70	54.52 ± 3.39	85.07 ± 1.59	82.62 ± 1.21	80.23 ± 1.96	80.75 ± 2.01	**88.97** ± 1.56
AID	[21, 27]	78.32 ± 0.46	48.39 ± 0.82	82.27 ± 0.46	81.02 ± 0.63	85.48 ± 0.52	73.93 ± 1.25	**87.99** ± 0.55
Pets	[25, 33]	88.08 ± 0.71	79.10 ± 1.85	88.64 ± 0.74	88.59 ± 0.78	90.25 ± 0.61	88.96 ± 0.16	**90.36** ± 0.74
Resisc45	[31, 40]	71.48 ± 0.43	47.16 ± 0.91	78.75 ± 0.59	76.07 ± 0.60	78.87 ± 0.48	66.59 ± 0.97	**85.35** ± 0.30
DTD	[32, 42]	58.66 ± 0.90	51.82 ± 0.84	61.75 ± 0.58	60.72 ± 0.84	63.83 ± 1.23	58.45 ± 0.84	**65.40** ± 1.06
CIFAR100	[70, 90]	56.97 ± 0.24	21.16 ± 0.74	61.69 ± 0.58	62.14 ± 0.36	68.87 ± 0.70	66.46 ± 0.48	**70.85** ± 0.50
Food	[70, 90]	55.67 ± 0.48	29.20 ± 0.33	61.51 ± 0.44	61.13 ± 0.42	69.50 ± 0.40	45.81 ± 0.78	**70.29** ± 0.36
Caltech101	[70, 90]	86.04 ± 0.76	83.30 ± 0.52	87.85 ± 1.30	87.40 ± 1.13	89.23 ± 1.10	89.31 ± 0.89	**90.40** ± 1.00
Flowers	[71, 91]	84.33 ± 0.54	71.63 ± 1.17	87.86 ± 0.55	87.24 ± 0.46	91.27 ± 0.58	82.91 ± 0.74	**94.48** ± 0.18
CUB2011	[140, 180]	61.84 ± 0.21	49.23 ± 0.61	66.88 ± 0.31	66.31 ± 0.17	70.88 ± 0.39	58.22 ± 0.52	**71.43** ± 0.23

Table 4: Average accuracy (%) on user tasks covering partial classes in image scenarios. Remaining details are consistent with Table 2.

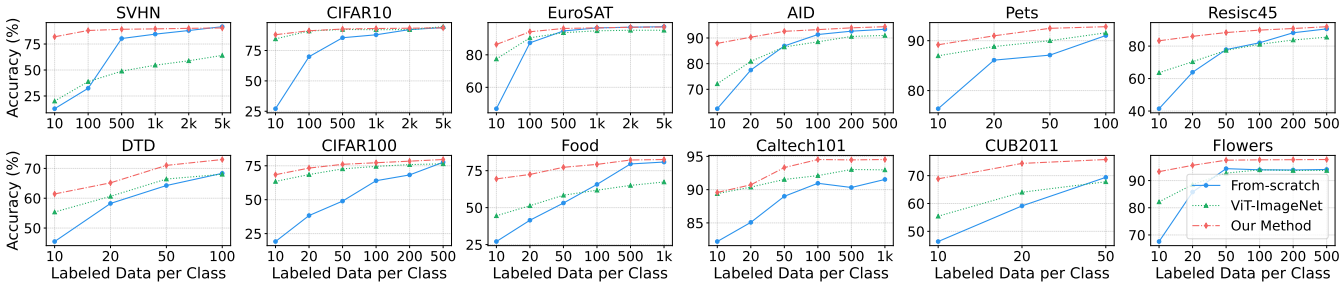

Figure 4: Average accuracy (%) on user tasks covering all classes in image scenarios, as the amount of labeled data per class increases.

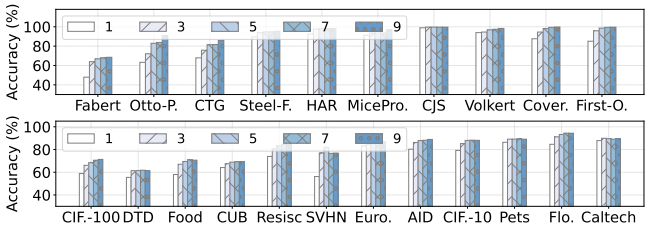

Figure 5: Average accuracy (%) on user tasks covering all classes in tabular and image scenarios, with 10 labeled instances per class, as the number of identified learnwares varies among $\{1, 3, 5, 7, 9\}$.

inference attacks [Lei *et al.*, 2024]. To support effective identification from numerous learnwares, an anchor-based mechanism [Xie *et al.*, 2023] enables efficient identification, and an evolvable learnware specification [Liu *et al.*, 2024] continuously enhances learnware characterization and identification as the system scales. For heterogeneous feature spaces, algorithms for searching and reusing learnwares are developed by learning a unified specification space [Tan *et al.*, 2023; Tan *et al.*, 2024a]. Recently, the first learnware dock system, Beimingwu [Tan *et al.*, 2024b], was released, providing implementations for the entire process of learnware paradigm.

For heterogeneous label spaces, prior work [Guo *et al.*, 2023] focuses on identifying a single learnware and requires a powerful public feature extractor, limiting its applicability especially in tabular scenarios. In contrast, our work makes the first attempt to identify and reuse effective learnware combinations across different label spaces, enabling broader applications of learnwares beyond their original purposes.

Utilizing Given Source Task(s) or Model(s). Domain adaptation [Ben-David *et al.*, 2006], transfer learning [Pan and Yang, 2010], and model reuse [Zhao *et al.*, 2020] focus on solving target tasks using given source task(s) or model(s). Several studies also explore heterogeneous label spaces, including disjoint label space transfer learning [Luo *et al.*, 2017], open set domain adaptation [Saito *et al.*, 2018], and partial domain adaptation [Cao *et al.*, 2019]. While these fields assume the given models are helpful for the target task or require access to raw source data, the learnware paradigm differs significantly, as it aims to identify and assemble useful models from numerous ones in a data-preserving way.

Model Pools. Model pools and hubs, like Hugging Face, have rapidly grown, hosting over a million models managed with only semantic descriptions. Some works, such as HuggingGPT [Shen *et al.*, 2023] and ToolLLM [Qin *et al.*, 2024], attempt to identify models or tools via these descriptions. However, since machine learning models are functions mapping inputs to outputs, statistical specifications in the learnware paradigm are essential for capturing their implicit capabilities and enabling effective identification and reuse beyond their original purposes. Some studies also focus on assessing the reusability or transferability of pre-trained models without fine-tuning [You *et al.*, 2021; Ding *et al.*, 2022; Zhang *et al.*, 2023], but these often require running all models on user data without considering data privacy, which is impractical in real-world scenarios with numerous models.

6 Conclusion

This paper presents the first attempt to identify and assemble helpful learnware combinations for user tasks across different label spaces without leaking raw data. To achieve this, we characterize model capabilities across classes and establish relationships between diverse label spaces, proposing a practical identification and reuse method to solve user tasks involving previously unseen label spaces. Extensive empirical evaluations validate the effectiveness of our approach.

Acknowledgments

This research was supported by NSFC (62250069) and the Collaborative Innovation Center of Novel Software Technology and Industrialization. The authors would like to thank Jia-Wei Shan and Tian Qin for helpful discussions, and Zi-Chen Zhao for his participation in preliminary experiments. We are also grateful to the anonymous reviewers for their helpful comments.

References

[Anguita *et al.*, 2013] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Proceedings of the 21st European Symposium on Artificial Neural Networks*, pages 437–442, 2013.

[Ayres-de Campos *et al.*, 2000] Diogo Ayres-de Campos, Joao Bernardes, Antonio Garrido, Joaquim Marques-de Sa, and Luis Pereira-Leite. Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, 9(5):311–318, 2000.

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

[Ben-David *et al.*, 2006] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19*, pages 137–144, 2006.

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

[Bossard *et al.*, 2014] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 - mining discriminative components with random forests. In *Proceedings of the 13th European Conference on Computer Vision*, pages 446–461, 2014.

[Bridge *et al.*, 2014] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving - learning to select a good heuristic. *Journal of automated reasoning*, 53(2):141–172, 2014.

[Camacho and Arron, 1995] Fernando Camacho and Geoffrey Arron. Effects of the regulators paclobutrazol and flurprimidol on the growth of terminal sprouts formed on trimmed silver maple trees. *Canadian Journal of Statistics*, 23(3):312–322, 1995.

[Cao *et al.*, 2019] Zhangjie Cao, Kaichao You, Mingsheng Long, Jianmin Wang, and Qiang Yang. Learning to transfer examples for partial domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2985–2994, 2019.

[Cheng *et al.*, 2017] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

[Cimpoi *et al.*, 2014] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

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

[Dosovitskiy *et al.*, 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[Edmonds and Karp, 1972] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

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

[Guyon *et al.*, 2015] Isabelle Guyon, Kristin Bennett, Gavin Cawley, Hugo Jair Escalante, Sergio Escalera, Tin Kam Ho, Núria Macià, Bisakha Ray, Mehreen Saeed, Alexander Statnikov, et al. Design of the 2015 chalearn automl challenge. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[Helber *et al.*, 2019] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[Higuera *et al.*, 2015] Clara Higuera, Katheleen J Gardiner, and Krzysztof J Cios. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PLOS ONE*, 10(6):e0129126, 2015.

[Hu *et al.*, 2022] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

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

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report, University of Toronto, 2009.

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

[Li *et al.*, 2004] Fei-Fei Li, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 178–178, 2004.

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

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

[Luo *et al.*, 2017] Zelun Luo, Yuliang Zou, Judy Hoffman, and Fei-Fei Li. Label efficient learning of transferable representations acrosss domains and tasks. In *Advances in Neural Information Processing Systems 30*, pages 165–177, 2017.

[Netzer *et al.*, 2011] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Precessing Systems 24 Workshops*, 2011.

[Nilsback and Zisserman, 2008] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the 6th Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729, 2008.

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

[Parkhi *et al.*, 2012] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012.

[Qin *et al.*, 2024] 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. ToolLLM: Facilitating large language models to master 16000+ real-world apis. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[Saito *et al.*, 2018] Kuniaki Saito, Shohei Yamamoto, Yoshitaka Ushiku, and Tatsuya Harada. Open set domain adaptation by backpropagation. In *Proceedings of the European Conference on Computer Vision*, pages 156–171, 2018.

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

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

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

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

[Tan *et al.*, 2024b] 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. Beimingwu: A learnware dock system. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5773–5782, 2024.

[Vanschoren *et al.*, 2013] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2013.

[Wah *et al.*, 2011] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[Winston, 2004] Wayne L Winston. *Operations research: applications and algorithm*. Thomson Learning, Inc., 2004.

[Wolpert, 1992] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

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

[Xia *et al.*, 2017] Gui-Song Xia, Jingwen Hu, Fan Hu, Baoguang Shi, Xiang Bai, Yanfei Zhong, Liangpei Zhang, and Xiaoqiang Lu. AID: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3965–3981, 2017.

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

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

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

[Zhao *et al.*, 2020] Peng Zhao, Le-Wen Cai, and Zhi-Hua Zhou. Handling concept drift via model reuse. *Machine Learning*, 109(3):533–568, 2020.

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

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

[Zhou, 2025] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. CRC Press, 2nd edition, 2025.

Supplementary Materials for "Identifying and Reusing Learnwares across Different Label Spaces"

Table of Contents

A Notations

The major notations used in this work are summarized in Table 5.

Category	Notation	Description
Developer	$D_i := (\boldsymbol{X}_i, \boldsymbol{y}_i)$	The local dataset of the i-th developer, defined over $\mathcal{X} \times \mathcal{Y}_i$.
	$\mathcal{X}, \mathcal{Y}_i$	The feature and label space unique to model f_i, respectively.
	(f_i, S_i)	The i-th learnware, where f_i is the model and S_i is the class-wise RKME specification.
	N	The number of learnwares in the learnware dock system.
User	$D_u := \boldsymbol{X}_u$	The unlabeled user dataset defined over the feature space $\mathcal{X}$.
	$\mathcal{Y}_u$	The label space of the user task, where $\mathcal{Y}_u \notin \{\mathcal{Y}_i\}_{i=1}^N$.
	m	The number of limited user labeled data for each class in $\mathcal{Y}_u$.
	$R_u := (S_u, \boldsymbol{w})$	User's task requirements, where S_u and $\boldsymbol{w}$ denote specification and estimated class probabilities.
Specification	n, k	The specification size and kernel function with reproducing kernel Hilbert space $\mathcal{H}_k$, respectively.
	$\hat{\mu}_c, \tilde{\mu}_c$	The empirical KME and RKME of class c, respectively.
	$S := \{(\boldsymbol{\beta}_c, \boldsymbol{Z}_c)\}_{c \in \mathcal{Y}}$	The class-wise RKME specification, where $\boldsymbol{\beta}_c \in \Delta^n$ and $\boldsymbol{Z}_c \in \mathcal{X}^n$.
Learnware Identification	$\{f_i \mid i \in I\}$	The helpful learnware combination returned by the learnware dock system for tackling the user task.
	M	The constraint for the size of learnware combinations, i.e., $\lvert I \rvert \leq M$.
	$\mathrm{sim}(S_{i,c_1}, S_{u,c_2})$	The similarity between class c_1 of the i-th learnware and class c_2 of the user task.
	λ	The constant which ensures non-negative similarity.
	$\mathcal{U}(I, R_u)$	The utility of a set of learnwares $\{f_i\}_{i \in I}$ to the user requirements R_u.

Table 5: Major notations used in this work.

B More Details of Our Approach

In this section, we provide more detailed explanation of the learnware identification method proposed in Section 3.2. Specifically, we first introduce the minimum-cost maximum-flow problem and then describe how the utility of a learnware combination to a user task is modeled as this problem.

B.1 Minimum-cost Maximum-flow Problem

The minimum-cost maximum-flow (MCMF) problem is a fundamental and classic problem in graph theory and network optimization. It serves as an extension of the maximum flow problem by incorporating an economic aspect, namely a cost associated with routing flow through each edge in the network. The goal is to achieve the maximum possible flow from a source to a sink, and among all possible ways to achieve this maximum flow, to select the one that incurs the minimum total cost.

Formally, we define a flow network as a directed graph $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of edges (or arcs). Within this network, two special vertices are designated: a source vertex $s \in V$, from which flow originates, and a sink vertex $t \in V$, where flow terminates. Each edge $(v, w) \in E$ is characterized by three attributes:

- **Capacity** $u(v, w) > 0$: The maximum amount of flow that can pass through the edge.
- **Cost** $c(v, w)$: The cost incurred per unit of flow sent along the edge. This cost can be positive, negative, or zero, though it's often non-negative in many practical applications.
- **Flow** $f(v, w)$: The actual amount of flow traversing the edge.

A valid flow f in the network must satisfy the following constraints: (1) For every edge $(v, w) \in E$, the flow must be non-negative and not exceed its capacity, i.e., $0 \leq f(v, w) \leq u(v, w)$; (2) For every vertex $v \in V \setminus \{s, t\}$ (i.e., any vertex other than the source or sink), the total flow entering the vertex must equal the total flow leaving it, i.e.,

$$\sum\nolimits_{(u,v)\in E} f(u, v) = \sum\nolimits_{(v,w)\in E} f(v, w).$$

The total cost of a given flow f is calculated by summing the costs incurred on all edges: $C(f) = \sum_{(v,w)\in E} f(v, w) \cdot c(v, w)$. The problem then is to find a flow f that first maximizes the total flow $F = \sum_{(s,v)\in E} f(s, v) - \sum_{(v,s)\in E} f(v, s)$ (the net flow out of the source), and second, among all such maximum flows, minimizes the total cost $C(f)$.

To further illustrate this problem, consider the following typical example [Winston, 2004]. The objective in this example is to maximize the number of cars passing through a network of roads (defined by nodes and arcs) between two points, namely the start and finish nodes. Each road (arc) within this network has a designated maximum capacity and a specific traversal time. The problem is to determine the maximum number of cars that can travel between these two points while minimizing the total traversal time taken. In this context, time is seen as the cost.

B.2 Discussion on the Modeling for the Utility of Learnware Combinations

As detailed in Section 3.2, we identify beneficial learnware combinations $\{f_i\}_{i\in I}$ for user tasks across heterogeneous label spaces. A core part is quantifying the utility $\mathcal{U}(I, R_u)$ of a combination I given user requirements $R_u = (S_u, \boldsymbol{w})$, where S_u is the user's class-wise RKME specification and $\boldsymbol{w}$ are estimated user class probabilities.

This utility is conceptualized as finding a maximum weight matching between learnware classes in I and user task classes in $\mathcal{Y}_u$. As defined in Section 3.2, the edge weight between a learnware class $c_1 \in \mathcal{Y}_i$ (from $f_i \in I$) with user class $c_2 \in \mathcal{Y}_u$ is $w_{c_2} \cdot \text{sim}(S_{i,c_1}, S_{u,c_2})$. The similarity $\text{sim}(S_{i,c_1}, S_{u,c_2})$ is formally defined as:

$$\text{sim}(S_{i,c_1}, S_{u,c_2}) = \lambda - \left\| \sum\nolimits_{j=1}^{n} \beta_{c_1,j} k(\boldsymbol{z}_{c_1,j}, \cdot) - \sum\nolimits_{j=1}^{n} \beta_{c_2,j} k(\boldsymbol{z}_{c_2,j}, \cdot) \right\|_{\mathcal{H}_k}^2,$$

where the latter term represents the MMD between class-wise RKME specifications S_{i,c_1} and S_{u,c_2} in the RKHS $\mathcal{H}_k$, and λ is a constant ensuring non-negative similarity. The matching is subject to two key constraints from Section 3.2:

1. Each class c_1 from a learnware f_i can be matched to at most one user class c_2 (i.e., $\sum_{c_2\in\mathcal{Y}_u} e^{(i)}_{c_1,c_2} \leq 1$).
2. Each user class c_2 can be matched by at most K learnware classes, acknowledging that each user class may have multiple similar or relevant learnware classes (i.e., $\sum_{i\in I, c_1\in\mathcal{Y}_i} e^{(i)}_{c_1,c_2} \leq K$).

Here, $e^{(i)}_{c_1,c_2} \in \{0, 1\}$ is a binary variable indicating whether class c_1 of learnware f_i is matched to user class c_2. To find the maximum weight matching, we transform this maximum weight matching problem into an equivalent minimum-cost maximum-flow (MCMF) problem. Explicitly, the MCMF network illustrated in Figure 2 is constructed as:

- A global source node (s) and a global sink node (t) are introduced.
- For each learnware f_i in the considered combination I, and for each of its classes $c_1 \in \mathcal{Y}_i$, a learnware class node representing (i, c_1) is created. An edge is added from s to each such node (i, c_1) with a capacity of 1 and a cost of 0. This ensures that each specific class from a learnware is used at most once in the matching.
- For each class $c_2 \in \mathcal{Y}_u$ in the user's task, a user class node c_2 is created. An edge is added from each user class node c_2 to the sink node t with a capacity of K and a cost of 0. This enforces the constraint that each user class can be matched by, or receive flow from, at most K learnware classes.
- For each pair consisting of a learnware class node (i, c_1) and a user class node c_2, an edge is added from (i, c_1) to c_2. This edge has a capacity of 1 (meaning this specific pairing can contribute at most one unit of flow) and a cost equal to the negated similarity weight: $-w_{c_2} \cdot \text{sim}(S_{i,c_1}, S_{u,c_2})$. Using negated weights allows us to employ a minimum-cost algorithm to achieve a maximum-weight matching.

Once this network is constructed, this problem can be effectively solved with the successive shortest path algorithm [Edmonds and Karp, 1972]. The utility $\mathcal{U}(I, R_u)$ of the learnware combination I is then precisely the negative of this calculated minimum cost. This MCMF formulation provides an effective way to quantify the utility of a given set of learnwares for the user's task, facilitating the identification of beneficial learnware combinations across heterogeneous label spaces.

C Proofs

C.1 Proof of Proposition 1

Proof of Proposition 1. For all $c \in \mathcal{Y}$, let $\hat{\mu}_c = \frac{1}{m_c}\sum_{i=1}^{m_c} k(\boldsymbol{x}_{c,i}, \cdot)$ denote the empirical KME of class c and $\tilde{\mu}_c = \sum_{j=1}^{n} \beta_{c,j} k(\boldsymbol{z}_{c,j}, \cdot)$ denote the RKME of class c. To analyze the convergence rate of $\tilde{\mu}_c$ to $\hat{\mu}_c$, we use the kernel herding method [Bach *et al.*, 2012], which generates a weighted mimic dataset $\{(\beta_{c,j}^{\text{mic}}, \boldsymbol{z}_{c,j}^{\text{mic}})\}_{j=1}^{n}$, where $\boldsymbol{\beta}_{\boldsymbol{c}}^{\text{mic}} \in \Delta^n$, to approximate $\hat{\mu}_c$. According to [Bach *et al.*, 2012], assuming $\sup_{\boldsymbol{x}\in\mathcal{X}} k(\boldsymbol{x}, \boldsymbol{x}) \leq B_{\mathcal{H}}$, we have:

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

where $\tilde{\mu}_c^{\text{mic}} = \sum_{j=1}^{n} \beta_{c,j}^{\text{mic}} k(\boldsymbol{z}_{c,j}^{\text{mic}}, \cdot)$. 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 $\tilde{\mu}_c$ to $\hat{\mu}_c$ can be bounded as follows:

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

which completes the proof. □

C.2 Proof of Theorem 1

Proof of Theorem 1. The analysis of time complexity for learnware identification in our approach can be divided into two parts: the first involves calculating the utility $\mathcal{U}(I, R_u)$, and the second optimizes Eq. (3) to determine the learnware combination.

We begin by analyzing the time complexity of the utility calculation. In Section 3.2, we model the utility of a learnware combination to a user task as a minimum-cost maximum-flow problem. To compute the utility, we employ the successive shortest path algorithm [Edmonds and Karp, 1972], which iteratively finds the augmenting path with the minimum cost per unit flow in a graph $G = (V, E)$ until no augmenting paths remain, resulting in a time complexity of $\mathcal{O}\left(F|E|\log|V|\right)$, where F denotes the maximum flow value. Without loss of generality, we assume that the maximum size of the learnware label spaces $\max_{i\in[N]}|\mathcal{Y}_i| \leq \mathcal{C}_f$, the size of the user task label space $|\mathcal{Y}_u| = \mathcal{C}_u$, and the learnware combination is denoted by I. In our modeling of the utility shown in Figure 2, the graph G consists of no more than $|V| = |I|\mathcal{C}_f + \mathcal{C}_u + 2$ nodes and $|E| = |I|\mathcal{C}_f\mathcal{C}_u + |I|\mathcal{C}_f + \mathcal{C}_u$ edges. The maximum flow F is less than $K\mathcal{C}_u$. Thus, the time complexity of computing the utility $\mathcal{U}(I, R_u)$ is $\mathcal{O}\left(K|I|\mathcal{C}_f\mathcal{C}_u^2\log\left(|I|\mathcal{C}_f + \mathcal{C}_u\right)\right)$. In Section 2, the size of the learnware combination I is constrained by M, such that $|I| \leq M$. Given that both M and K are constants and are typically small (less than or equal to 5 in experiments), the time complexity can be restated as $\mathcal{O}\left(\mathcal{C}_f\mathcal{C}_u^2\log\left(\mathcal{C}_f + \mathcal{C}_u\right)\right)$.

Since a greedy algorithm is employed to iteratively optimize Eq. (3), at the t-th iteration, the learnware combination $I_t = I_{t-1} \cup \{i\}$ is derived by solving Eq. (4). The time complexity for each iteration is $\mathcal{O}\left(N\mathcal{C}_f\mathcal{C}_u^2\log\left(\mathcal{C}_f + \mathcal{C}_u\right)\right)$, where N denotes the number of available learnwares. According to Algorithm 2, the maximum iteration number is M, which is a constant and is typically small (less than or equal to 5 in experiments). Thus, the time complexity of our learnware identification method is $\mathcal{O}\left(N\mathcal{C}_f\mathcal{C}_u^2\log\left(\mathcal{C}_f + \mathcal{C}_u\right)\right)$, which completes the proof. □

D Additional Details and Results for Experiments

In this section, we first provide omitted details of the experimental setup. Then, we conduct additional experiments, including comparing our method with full parameter tuning and LoRA [Hu *et al.*, 2022], exploring the relationship between the amount of labeled data and learnware combinations, and analyzing the parameter stability and robustness of our approach.

D.1 Omitted Details about Experimental Setup

Tabular Datasets. The tabular scenarios presented in experiments are derived from 10 real-world datasets: Covertype [Blackard and Dean, 1999], HAR [Anguita *et al.*, 2013], Otto-Product [Vanschoren *et al.*, 2013], MiceProtein [Higuera *et al.*, 2015], CJS [Camacho and Arron, 1995], Steel-Fault [Vanschoren *et al.*, 2013], First-Order [Bridge *et al.*, 2014], CTG [Ayres-de Campos *et al.*, 2000], Fabert [Guyon *et al.*, 2015], and Volkert [Guyon *et al.*, 2015]. These datasets are accessible on the OpenML platform [Vanschoren *et al.*, 2013] and cover a wide range of domains such as healthcare, ecology, business, informatics, industry, and biology. For each tabular dataset, details including its OpenML ID and the number of classes are provided in Table 6.

Image Datasets. The image scenarios presented in experiments are derived from 12 real-world datasets: SVHN [Netzer *et al.*, 2011], CIFAR10 [Krizhevsky and Hinton, 2009], EuroSAT [Helber *et al.*, 2019], AID [Xia *et al.*, 2017], Pets [Parkhi *et al.*, 2012], Resisc45 [Cheng *et al.*, 2017], DTD [Cimpoi *et al.*, 2014], CIFAR100 [Krizhevsky and Hinton, 2009], Food [Bossard *et al.*, 2014], Caltech101 [Li *et al.*, 2004], Flowers [Nilsback and Zisserman, 2008], and CUB2011 [Wah *et al.*, 2011]. These public datasets encompass a variety of domains, including plants, animals, food, remote sensing, and objects, ensuring broad applicability across various areas. For each image dataset, details about the number of classes are presented in Table 7. Example instances from these datasets are illustrated in Figure 6.

Dataset	#Classes	OpenML ID	Dataset	#Classes	OpenML ID
HAR [Anguita *et al.*, 2013]	12	1478	Covertype [Blackard and Dean, 1999]	7	150
Volkert [Guyon *et al.*, 2015]	10	41166	MiceProtein [Higuera *et al.*, 2015]	8	40966
CJS [Camacho and Arron, 1995]	6	23380	Steel-Fault [Vanschoren *et al.*, 2013]	7	40982
First-Order [Bridge *et al.*, 2014]	6	1475	CTG [Ayres-de Campos *et al.*, 2000]	10	1466
Fabert [Guyon *et al.*, 2015]	7	41164	Otto-Product [Vanschoren *et al.*, 2013]	9	45548

Table 6: The number of classes and OpenML ID for each tabular dataset.

Dataset	#Classes	Dataset	#Classes
SVHN [Netzer *et al.*, 2011]	10	DTD [Cimpoi *et al.*, 2014]	47
CIFAR10 [Krizhevsky and Hinton, 2009]	10	CIFAR100 [Krizhevsky and Hinton, 2009]	100
EuroSAT [Helber *et al.*, 2019]	10	Food [Bossard *et al.*, 2014]	101
AID [Xia *et al.*, 2017]	30	Caltech101 [Li *et al.*, 2004]	101
Pets [Parkhi *et al.*, 2012]	37	Flowers [Nilsback and Zisserman, 2008]	102
Resisc45 [Cheng *et al.*, 2017]	45	CUB2011 [Wah *et al.*, 2011]	200

Table 7: The number of classes for each image dataset.

For each dataset, the training and test data are partitioned according to their original configurations; if none are available, they are randomly divided in a 4:1 ratio. In the experiments, the training data are employed to train models that form the learnwares, while the test data are regarded as unlabeled user data for subsequent evaluation.

Model Configurations. In our experiments, models derived from tabular scenarios are trained using the LightGBM [Ke *et al.*, 2017] algorithm. We select the hyperparameter combinations of (learning rate, number of leaves, maximum depth) from the set $\{(0.015, 224, 66), (0.005, 300, 50), (0.01, 128, 80), (0.15, 224, 80), (0.01, 300, 66)\}$. For image scenarios, models are trained with the ResNet50 [He *et al.*, 2016] model for 100 epochs using the SGD optimizer. The initial learning rate is selected from the set $\{0.1, 0.01, 0.001\}$. During the training process, a cosine annealing strategy is applied for managing the learning rate decay, facilitating the model to converge to a better solution.

D.2 Comparison with More Fine-Tuning Methods

In our work, we compare our method with fine-tuning the last linear layer (*ViT Linear Probing*) of the medium-to-large-scale model ViT-L/32 [Dosovitskiy *et al.*, 2021], which has 307 million parameters and is pretrained on ImageNet [Russakovsky *et al.*, 2015]. To further evaluate our method, we compare it with fine-tuning all model parameters (*ViT Fine-Tuning*) and fine-tuning using LoRA [Hu *et al.*, 2022] (*ViT LoRA*), an efficient parameter fine-tuning method. Specifically, we conduct additional experiments on user tasks covering all classes in image scenarios, where each user task includes 10 labeled instances per class. The results presented in Table 8 show that our method significantly outperforms these contenders.

Besides, since *ViT Linear Probing* and *ViT Fine-Tuning* perform better than *ViT LoRA*, we further compare our method with these two as the labeled data per class increases from 10 to 5,000. If the available data per class is less than 5,000, we include all of it. As shown in Figure 7, our method's performance improves as the labeled data increase, and in most scenarios, it is superior to both *ViT Linear Probing* and *ViT Fine-Tuning*, even when all labeled data are accessible. This implies that reusing helpful learnware combinations could be more effective even when sufficient labeled data are available.

D.3 Relationship between the Amount of Labeled Data and Learnware Combinations

To determine whether using 10 examples per class is accurate enough to measure the similarity between models and user tasks, we explore the relationship between the amount of labeled data and learnware combinations. Since more labeled data leads to better model reuse performance, to ensure precise comparisons, we fix the reuse data at 10 examples per class and varied the data for generating specifications with 10, 50, and 200 examples per class. The results in Table 9 show that more data has slightly improved search performance, with 10 examples being sufficient to measure the similarity between models and tasks.

D.4 Analysis of the Parameter Stability and Robustness

We further evaluate the parameter stability and robustness of our approach. In Section 3.2, since each user class may have multiple similar learnware classes, we impose a constraint that each user class can be matched by at most K learnware classes, as illustrated in Figure 2. In our experiments, only the hyperparameter K is adjusted during the learnware identification process. This raises questions regarding the parameter stability of our approach as the hyperparameter K varies.

To investigate the robustness of our approach, we present results for both tabular and image scenarios with the hyperparameter K varying among the set $\{1, 2, 3, 4\}$. To better illustrate the results, we select the *From-scratch* method as the baseline, as shown in Figures 8 and 9. The empirical results indicate that as the hyperparameter K changes, the performance of our approach does not fluctuate significantly, highlighting the parameter stability of our method. Additionally, for all $K \in \{1, 2, 3, 4\}$, our approach consistently outperforms the baseline *From-scratch*, further demonstrating the robustness of our method.

Figure 6: Example instances from each image dataset.

Method	CIFAR10	AID	Pets	EuroSAT	Resisc45	DTD	SVHN	CIFAR100	Food	Caltech101	Flowers	CUB2011
ViT Lora	74.92	78.67	19.65	71.25	81.66	59.02	48.30	45.77	84.62	26.91	72.28	28.03
ViT Linear Probing	85.44	78.50	20.15	71.92	87.32	63.79	55.05	63.65	88.47	43.64	81.56	55.31
ViT Fine-Tuning	84.43	78.15	20.24	72.30	86.02	64.79	53.88	59.15	86.98	43.68	83.70	51.93
Our Method	**86.20**	**87.86**	**78.19**	**87.18**	**89.28**	**83.42**	**62.12**	**68.16**	**89.38**	**68.49**	**93.83**	**68.98**

Table 8: Average accuracy (%) on user tasks covering all classes across 12 image scenarios. Each user task includes 10 labeled instances per class. The best results are emphasized in bold.

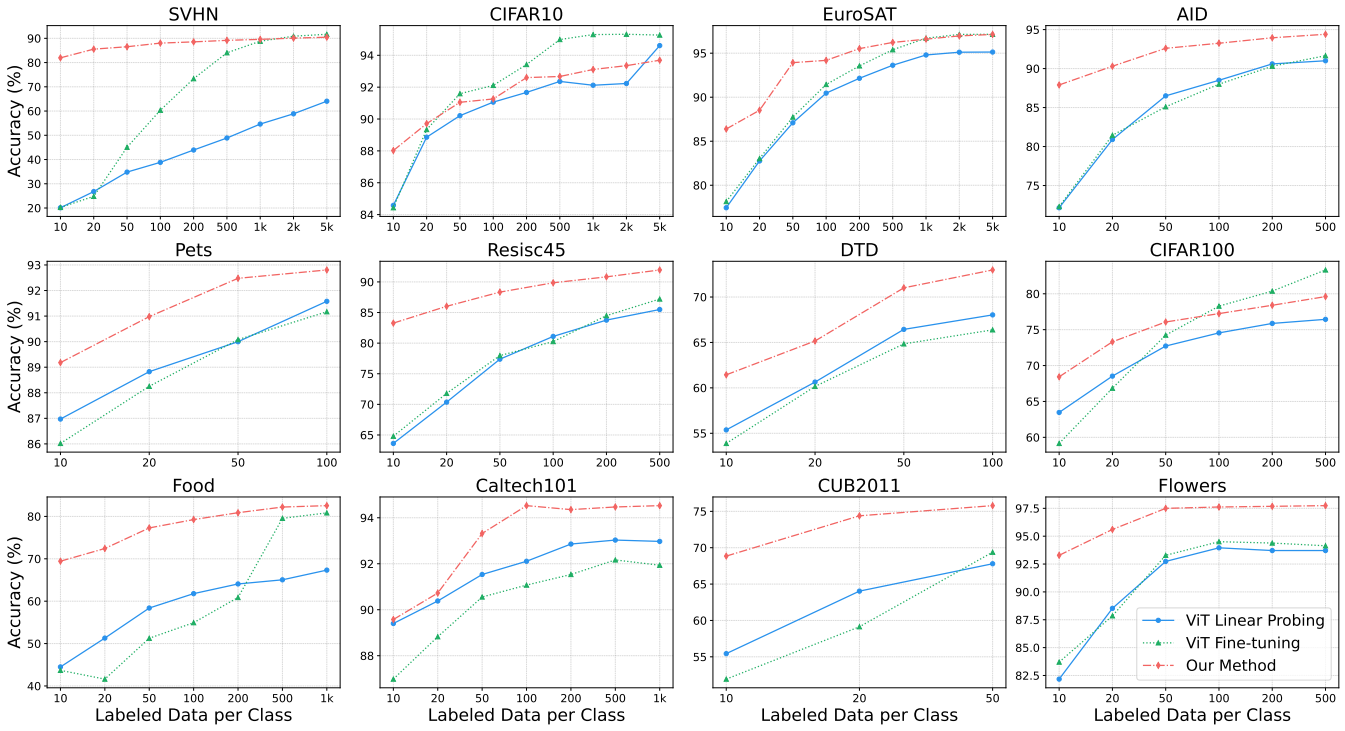

Figure 7: Average accuracy (%) on user tasks covering all classes in 12 image scenarios as the labeled data per class increase from 10 to 5,000. If the available data per class is less than 5,000, we include all of it. The remaining experimental setup is consistent with Figure 4.

Data per Class	CIFAR10	AID	Pets	EuroSAT	Resisc45	DTD	SVHN	CIFAR100	Food	Caltech101	Flowers	CUB2011
10	86.20	87.18	89.28	87.86	83.42	62.12	78.19	68.16	68.49	89.38	93.83	68.98
50	88.07	87.90	89.28	88.00	83.42	62.12	82.21	68.16	69.36	89.57	93.83	68.98
200	89.27	87.10	89.28	87.59	83.42	62.12	76.61	68.16	68.91	89.00	93.83	68.98

Table 9: Average accuracy (%) on user tasks covering all classes across 12 image scenarios, with the amount of labeled data for generating specifications increasing and the data for reusing models fixed at 10 instances per class.

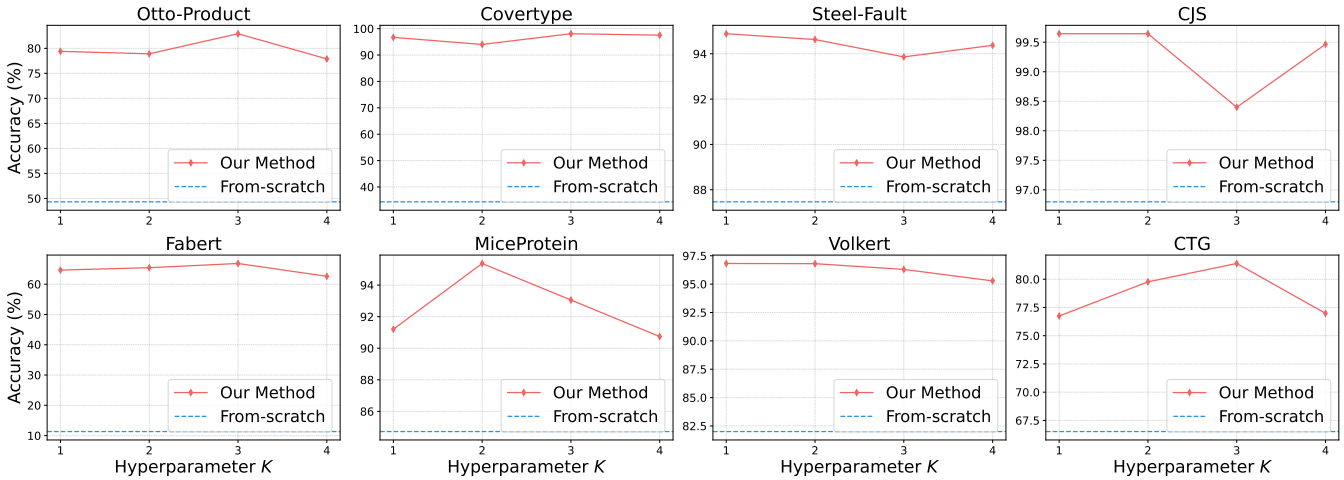

Figure 8: The average classification accuracy (%) on user tasks covering all classes in tabular scenarios, as the hyperparameter K varies among the set $\{1, 2, 3, 4\}$. Each user task includes 10 labeled instances per class.

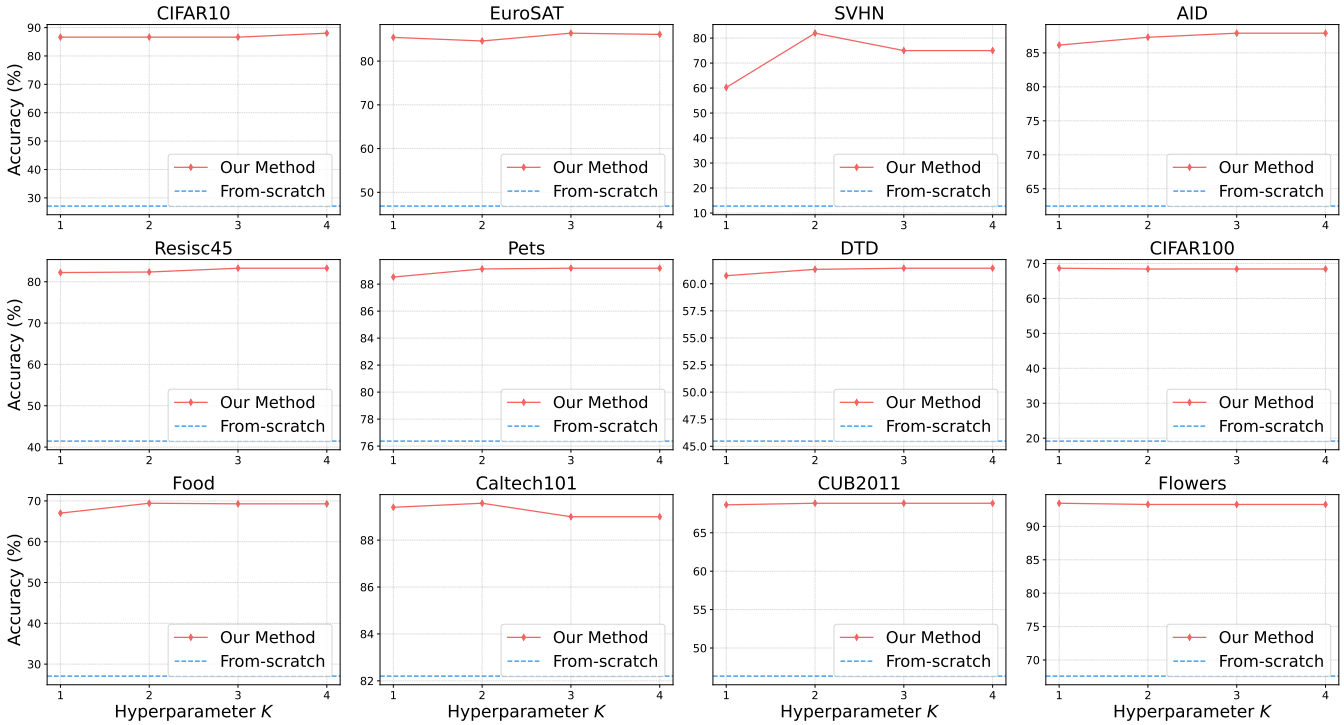

Figure 9: The average classification accuracy (%) on user tasks covering all classes in image scenarios, as the hyperparameter K varies among the set $\{1, 2, 3, 4\}$. Each user task includes 10 labeled instances per class.