Cost-Effective and Interpretable Job Skill Recommendation with Deep Reinforcement Learning

Ying Sun[1,2,3†], Fuzhen Zhuang[1,3*], Hengshu Zhu[2*], Qing He[1,3], Hui Xiong[4]
[1]Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China [2]Baidu Talent Intelligence Center, Baidu Inc., Beijing, 100085, China
[3]University of Chinese Academy of Sciences, Beijing 100049, China
[4]Rutgers, the State University of New Jersey, Newark, USA
{sunying17g, zhuangfuzhen, heqing}@ict.ac.cn, zhuhengshu@baidu.com, hxiong@rutgers.edu

ABSTRACT

Nowadays, as organizations operate in very fast-paced and competitive environments, workforce has to be agile and adaptable to regularly learning new job skills. However, it is nontrivial for talents to know which skills to develop at each working stage. To this end, in this paper, we aim to develop a cost-effective recommendation system based on deep reinforcement learning, which can provide personalized and interpretable job skill recommendation for each talent. Specifically, we first design an environment to estimate the utilities of skill learning by mining the massive job advertisement data, which includes a skill-matching-based salary estimator and a frequent itemset-based learning difficulty estimator. Based on the environment, we design a Skill Recommendation Deep Q-Network (SRDQN) with multi-task structure to estimate the long-term skill learning utilities. In particular, SRDQN recommends job skills in a personalized and cost-effective manner; that is, the talents will only learn the recommended necessary skills for achieving their career goals. Finally, extensive experiments on a real-world dataset clearly validate the effectiveness and interpretability of our approach.

KEYWORDS

Reinforcement Learning, Skill Recommendation, Data Mining

ACM Reference Format:
Ying Sun, Fuzhen Zhuang, Hengshu Zhu, Qing He, Hui Xiong. 2021. Cost-Effective and Interpretable Job Skill Recommendation with Deep Reinforcement Learning. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia* ACM, NY, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449985

1 INTRODUCTION

In the world of VUCA (Volatility, Uncertainty, Complexity, and Ambiguity), talents are required to possess more specialized job skills to meet the reality of increasingly refined division of labour and higher levels of work intensification. As shown in recent studies, owning the advanced job skills can help talents significantly bring up their salary in the labour market [5, 9]. Therefore, learning new skills becomes essential for talents to keep up with the fast-moving technical trend, which diversifies their job options and builds competitive advantage. However, due to the dynamic nature and the uncertain benefit of skill learning, there still lacks a sustainable guidance for talents on learning the right and proper job skills.

During the past decades, large efforts have been devoted to skill recommendation for talents [2, 8, 19]. Nevertheless, these works mainly focus on measuring the relevance of job skills to the current occupations of talents for making recommendations, while the multi-faceted skill learning utilities, such as the learning cost and the potential benefit, have been largely neglected. Meanwhile, existing skill recommendation approaches usually perform in a static manner instead of seizing the long-term influence of skills on career development. Indeed, multiple concerns should be considered when choosing the right and proper skills to learn. Ideally, the chosen skills should bring long-term benefits and have an acceptable learning cost. However, quantifying these factors and make a comprehensive decision is a non-trivial task. There are two-fold challenges along this line. First, since skills have complex interactions, the utilities may differ for talents owning different skill sets. Second, since there are an exponentially large amount of possible skill combinations on the learning path, the long-term benefit of skill learning is abstract and hard to be modeled. For example, although skills required by highly-paid jobs may bring high long-term benefit, the learning path towards the ideal job may be impracticably difficult to approach.

To this end, in this paper, we introduce a new research problem, namely *sustainable job skill recommendation*, which aims to provide long-sighted, cost-effective, and explainable recommendations for talents who want to learn new job skills. To address this problem, we propose a data-driven recommendation framework based on the technology of deep reinforcement learning (RL). Specifically, we first design an environment to estimate the utilities of skill learning by mining the massive job advertisement data, including a skill-matching-based salary estimator and a frequent itemset-based learning difficulty estimator. Based on the environment, we propose a multi-objective RL formulation for the skill-learning process to model the long-term effect. Then, we specially design a novel Skill Recommendation Deep Q-Network (SRDQN) with multi-task structure to estimate the long-term skill learning utilities. SRDQN

*Fuzhen Zhuang and Hengshu Zhu are corresponding authors.
†This work was accomplished when the first author working as an intern in Baidu supervised by the third author.

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449985

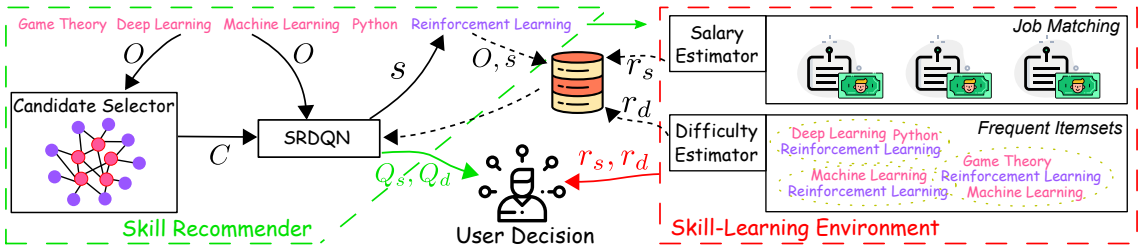


Figure 1: Framework overview.

separately models long-term salary and difficulty utility to provide the users with personalized skill recommendations, which effectively auxiliary human decisions in an interpretable way. In particular, SRDQN can recommend job skills in a personalized and cost-effective manner, which helps talents to gain more salary while pay less learning efforts considering their owned skills. Finally, we conduct extensive experiments on a real-world dataset. The experimental results clearly validate the effectiveness and interpretability of our approach.

2 PRELIMINARIES

In this section, we introduce the preliminaries, including problem formulation, data description, and framework overview.

2.1 Problem Formulation and Data Description

In this paper, we introduce a sustainable skill recommendation problem. Specifically, given the skill set owned by a talent, denoted by $O = \{o_1, o_2, \cdots\}$, we aim to recommend an ordered sequence of to-be-learned skills that continuously bring high profit to the talent with low learning cost. In this work, we regard profit as salary increase and cost as learning difficulty. In particular, we apply online job advertisement data to achieve quantitative estimation on salary increase and learning difficulty. Then, we train a skill recommendation model based on deep RL.

The job posting data of this paper were collected from a popular Chinese online recruitment website, namely Lagou[1]. Our data contains job postings across a period of 36 months ranging from 2016.07 to 2019.06. Each job posting consists of the offered job salary and a descriptive text introducing job skill requirements. In order to mine the interaction among skills, we formulate the job advertisement as $J^j = (R^j, m^j)$, where m^j denotes the job salary and $R^j = \{r_1^j, r_2^j, \cdots r_k^j\}$ denotes the required skill set.

2.2 Framework Overview

Figure 1 shows the overall framework of this paper. Specifically, we first design a skill-learning environment to quantify the skill learning utilities. Then, we propose an RL-based skill recommender to provide comprehensive skill-learning references.

The skill-learning environment simulates the salary increase and learning difficulty by mining skill interaction from job advertisement data. The salary estimator (Section 3.1) is motivated by person-job fit theory [4, 20, 37], where fitness between individual ability and job requirement is considered to decide the competency between the talent and job. Therefore, it matches the talent skill set to jobs in the market and estimates the talent's expected salary according to these jobs' offered salary. The difficulty estimator is motivated by the theory of the learning transfer [1, 10], where owning related knowledge is considered to make learning a new skill easier. Therefore, the difficulty estimator (Section 3.2) calculates the skill's relevance with the current skill set and estimates the difficulty with the conditional probability. In particular, we develop a *frequent itemset*-based algorithm to accelerate the calculation.

Based on the environment, we formulate a multi-objective RL problem (Section 4.1), which regards the salary increase and learning difficulty as two kinds of rewards. Then, we propose a novel Skill Recommendation Deep Q-Network (SRDQN) to learn a comprehensive policy towards high long-term returns. SRDQN (Section 4.2) has a multi-task structure that separately estimates the long-term utilities of salary increase and learning difficulty. In this way, SRDQN can recommend skills in a highly explainable manner. Moreover, to reduce time complexity and enhance data-efficiency during training, a *candidate selector* (Section 4.3) is designed to reduce the number of skills considered for recommendation, which filters the skills that are densely connected with the talent's skills on a skill graph. During the training phase, SRDQN sequentially recommends skills for some initial skill sets and utilizes the reward information to update its parameters. To accelerate training, we adopt experience replay strategy [18]. Specifically, a cache database stores the recommendation record of each step (i.e., state, action, and reward). The model parameters will be updated with the cache database in an off-policy manner. During the using phase, SRDQN takes the user's skill set and estimates the short-term (reward) and long-term (Q-value) utilities of learning different skills. Then, it recommends skills and provides explainable utility information. According to the user's choice, SRDQN can adjust the subsequent recommendations to form the learning path.

3 SKILL LEARNING ENVIRONMENT

This section introduces the skill-learning environment. Specifically, we propose to estimate salary and difficulty based on job advertisement data and design fast implementations to raise the training efficiency of the RL model. The major notations of this paper are shown in Table 1.

3.1 Salary Estimator

3.1.1 Job Matching for Salary Estimation. In real-world scenarios, there may be complicated factors and situations that influence job salary. Therefore, accurately modeling job salary is challenging. In this paper, to avoid error accumulating and to assure the interpretability, we measure salary in a simplified way based on the general skill influence. Specifically, the main idea is to match the talent skill set (i.e., O) with the job requirement (i.e., R^j) to find out

[1] https://www.lagou.com/

Table 1: Major notations in this paper.

notation	Explanation
O^t	The skill set (state) at the t-th step.
o_i	The i-th element in skill set O.
s^t	The selected skill (action) at the t-th step.
J^j	The j-th job.
R^j	The skill requirement of the j-th job.
r_i^j	The i-th skill in R^j.
m^j	The offered salary of the j-th job.
$f_s(O)$	The salary estimation of skill set O.
$f_p(O, J^j)$	The probability of talent getting job J^j with O.
$f_e(O, J^j)$	The expected salary that O can get from J^j.
$f_d(s, O)$	The difficulty of learning skill s based on O.
w	Parameter of the sigmoid function.
θ_s	The skill coverage threshold for getting a job in the salary estimator.
$p(s\|O)$	The appearing probability of skill s given O.
$p(s)$	The probability of skill s appearing in job postings.
θ_d	The probability threshold for being a frequent itemset.
$\mathcal{F}^t$	The frequent subsets at the t-th step.
F_i^t	The i-th element in $\mathcal{F}^t$.
$r(s\|O)$	The reward of learning s under state O.
$r_s(s\|O)$	The salary increase of learning s under state O.
$r_d(s\|O)$	The difficulty of learning s under state O.
π	Policy.
$Q_\pi(s\|O)$	The Q-value of action s at state O under policy π.
$Q_\pi^s(s\|O)$	The salary Q-value of taking action s at state O under policy π.
$Q_\pi^d(s\|O)$	The difficulty Q-value of taking action s at state O under policy π.
γ	Discount factor.
α	Difficulty importance factor.
λ	Loss function parameter.
N_c	The candidate pool size.
$C(O)$	The candidate pool of state O.
N_s	The total number of skills.
N_J	The total number of job postings.

jobs that the talent is qualified for. In this way, we can estimate the talent salary according to the offered salary of these jobs. In this paper, we estimate the salary of a skill set O, denoted by $f_s(O)$, as the averaged top-K salary of the matched jobs. Intuitively, a talent is qualified for a job when all the skill requirements are satisfied. Formally, we can define a skill set O to be *hard-qualified* for job J^j if $R^j \subseteq O$. However, hard qualification can easily cause the non-smoothness problem, i.e., learning a new skill may cause a sudden change in salary, which brings difficulty for model convergence.

To alleviate this problem, we define *soft qualification*. Intuitively, the more a talent meets a job's requirement, the more possible they can get this job. We model the probability as $f_p(O, J^j) = (1 + \exp(-w(\frac{|R^j \cap O|}{|R^j|} - \frac{1}{2})))^{-1}$, where $w \in \mathbb{R}$ is a super-parameter. A large w assures $f_p(O, J^j)$ to be close to 1 when $R^j \subseteq O$ and close to 0 when $R^j \cap O = \emptyset$. Motivated by the real-world recruiting scenarios, we suppose that the talent cannot get the job when $\frac{|R^j \cap O|}{|R^j|}$ is less than a threshold θ_s. In this way, the expected salary that O can get from J^j is $f_e(O, J^j) = \mathbb{1}\{\frac{|R^j \cap O|}{|R^j|} \geq \theta_s\} m^j f_p(O, J^j)$. Then, we estimate $f_s(O)$ as the averaged top-K expected salary. Notably, the soft qualification degenerates into the hard qualification when θ_s is set to 1. Apart from its smoothness, soft qualification can also alleviate data noise. Specifically, jobs posted by different recruiters may differ in skill distributions. For example, some recruiters may describe inessential skill requirements, while others may not. In this case, despite denoting the same duty, jobs containing detailed skill requirements are harder to be matched with hard-qualification. With soft-qualification, the talent is regarded as partially qualified for the job as long as they meet most of the requirements, which loses the constraint and significantly alleviates the noise. Corresponding to the common sense that owning more skills leads to higher-paid jobs, f_s is monotonic. Specifically, for any O and $o' \notin O$, we have $\forall j \quad f_e(O, J^j) \leq f_e(O \cup \{o'\}, J^j)$. This means that, for each of the top-K jobs, the expected salary will not decrease, thus we have $f_s(O) \leq f_s(O \cup \{o'\})$.

Algorithm 1 Sequential Job Matching Algorithm

Input: $R^1, \cdots, R^{N_J}$: jobs formed as skill sets; $m^1, \cdots, m^{N_J}$: job salary; $o_1, \cdots, o_L$: input skill sequence; $topK$: number of top salary to be averaged; N_s: number of skills; θ_s: probability threshold; w: probability parameter;

Output: $salary_1, \cdots salary_L$: salary prediction for each step;

```
Initialize N_s empty lists S_1, S_2, ··· S_{N_s};
for j ← 1 ··· N_J do
    count_j ← 0;
    for each s ∈ R^j do
        S_s ← S_s ∪ {j};
heap ← empty max-heap ;
O' ← empty set;
for i ← 1 ··· L do
    for each j ∈ S_{o_i} do
        count_j ← count_j + 1;
        if count_j / |R^j| ≥ θ_s then
            Add (job = j, val = m^j / (1+exp(−w(count_j/|R^j| − 1/2)))) to heap;
    salary_i ← 0;
    storeList ← empty list;
    for k ← 1 ··· topK do
        x ← pop heap;
        if x.job ∉ storeList then
            storeList ← storeList ∪ {x.job};
            salary_i ← salary_i + x.val/K;
    Clear heap;
    for each j ∈ storeList do
        Add (job = j, val = m^j / (1+exp(−w(count_j/|R^j| − 1/2)))) to heap;
return salary_1, ··· salary_L;
```

3.1.2 Job Matching Algorithm. As we have mentioned before, our proposed recommendation model sequentially recommends skills and gets the reward of each step for parameter training. For each step, the salary estimator needs to find out the qualified jobs in all the N_J job postings and accordingly estimate the job salary. However, going through the job set will bring high computational complexity. Specifically, supposing we recommend L skills for each skill set, the time complexity reaches $O(ML)$, where $M = \sum_{j=1}^{N_J} |R^j|$.

To accelerate the matching process, we incrementally update the skill coverage status (i.e., the number of covered skills) of the jobs. We show this process in Algorithm 1. Specifically, we build a job set for each of the skills, where set S_i stores the jobs requiring the i-th skill. In this way, when learning a new skill i, we only update coverage of jobs in the corresponding job set S_i. The space requirement of this process is $O(M)$, which is the same as the job

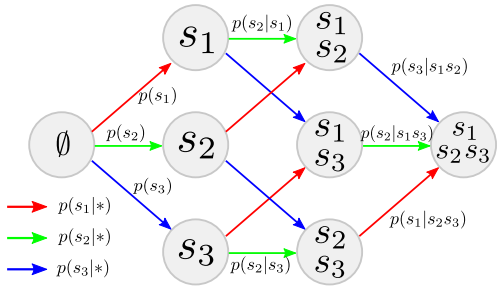


Figure 2: Probabilistic graph for frequent skill sets.

storage and raises no space complexity. Finally, we adopt a max-heap [7] to maintain the top-K salary of the matched jobs.

3.2 Learning Difficulty Estimator

To build the learning difficulty estimator, we first propose a relevance-based difficulty measurement and approximate it with frequent itemset to accelerate the estimation. Finally, we design a fast graph-based algorithm to reduce time complexity.

3.2.1 Relevance-based Difficulty Estimation. According to the theory of the transfer of learning [1, 10], the current skill set influences the difficulty of learning a new skill. For example, talents familiar with *"Java"* can more easily learn *"Scala"*. Motivated by this theory, we use the relevance among skills for difficulty estimation. Specifically, we formulate the difficulty measurement as $f_d(s, O) = -\max_{O_s \subseteq O} \log p(s|O_s)$, where $p(s|O_s)$ denotes the probability of s appearing in a job that requires O_s. With maximum likelihood estimation (MLE), we can estimate the probabilities as $p(s|O_s) = \frac{\sum_{j=1}^{N_J} \mathbb{1}\{\{s\} \cup O_s \subseteq R^j\}}{\sum_{j=1}^{N_J} \mathbb{1}\{O_s \subseteq R^j\}}$, $p(s) = \frac{\sum_{j=1}^{N_J} \mathbb{1}\{s \in R^j\}}{N_J}$. Notably, $p(s|\emptyset) = p(s)$. Corresponding to the fact that mastering more skills decreases the difficulty of learning a new skill, f_d is monotonic. Specifically, given any skill $a \notin O \cup \{s\}$, we have

$$f_d(s, O \cup \{a\}) = -\max_{O_s \subseteq O \cup \{a\}} \log p(s|O_s)$$
$$= -\max_{O_s \in O} \max\{\log p(s|O_s), \log p(s|O_s \cup \{a\})\} \leq f_d(s, O).$$

3.2.2 Frequent Itemset-based Approximation. To calculate $f_d(s, O)$, the estimator needs to go through all the possible subsets of O, which is intractable, for O has $2^{|O|} - 1$ subsets in total. To reduce time complexity, we propose to approximate the difficulty with frequent itemsets. Specifically, considering the fact that $p(s|O_s) = \frac{p(\{s\} \cup O_s)}{p(O_s)}$ is meaningless if O_s seldom appears in jobs, we approximate $f_d(s, O)$ by eliminating the infrequent subsets with a threshold θ_d. Formally, we approximate the difficulty as $f_d(s, O) = -\max_{O_s \subseteq O, p(O_s) \geq \theta_d} \log p(s|O_s)$.

As we have mentioned before, in our system, the estimator needs to calculate the learning difficulty of each skill in the sequence. With the property that all the subsets of a frequent itemset are also frequent, we can incrementally find out the frequent subsets of each step, written as $\mathcal{F}^t = \{F_0^t, F_1^t, \cdots\}$. Specifically, we add the new skill to each element in $\mathcal{F}^{t-1}$ and check if the new set is frequent. Formally, $\mathcal{F}^t = \mathcal{F}^{t-1} \cup \{F \cup \{s\} | F \in \mathcal{F}^{t-1}, p(F \cup \{s\}) > \theta_d\}$.

Algorithm 2 Difficulty Estimation Algorithm

Input: $F_0, F_1, \cdots, F_{N_f}$: a list of frequent skill sets, F_0 is the empty set; $prob_1, \cdots, prob_{N_f}$: the probability of the skill sets; $o_1, o_2, \cdots, o_L$: input skill sequence; θ_d: probability threshold; N_s: number of skills;
Output: $d_1, \cdots d_L$: difficulty estimation of each step;

```
Initialize empty sets Edges_0, ..., Edges_{N_f};
for j = 1 → N_f do
    for each s ∈ F_j do
        i ← index of frequent set F_j − {s};
        Add edge < start = i, end = j, skill = s > to Edges_i;
Initialize empty sets Active_1, ..., Active_{N_s};
for each edge ∈ Edges_0 do
    Add edge to Active_{edge.skill};
for each i = 1 → L do
    d_i = +∞;
    for each edge ∈ Active_{o_i} do
        d_i ← min(−log(prob_{edge.end} / prob_{e.start}), d_i);
        for each nextEdge ∈ Edges_{edge.end} do
            Add nextEdge to Active_{edge.skill};
return d_1, ··· d_L;
```

3.2.3 Graph-based Fast Difficulty Estimation. Though we have largely reduced the number of subsets, going through $\mathcal{F}^t$ still brings high time complexity. To solve this problem, we propose a graph-based difficulty estimation algorithm, which can calculate the difficulty and find out $\mathcal{F}^{t+1}$ without going through all the elements in $\mathcal{F}^t$. Specifically, we find all the possible frequent skill sets (including the empty set) in advance with *frequent itemset mining algorithm* [13]. Then, we organize the skill sets with a graph structure, formed like the example shown in Figure 2. In this graph, each node represents a frequent skill set, a directed edge $< start, end >$ exists between node *start* and node *end* if $|F_{end} - F_{start}| = 1$. Supposing s is the *incremented skill* on the edge (i.e., $\{s\} \cup F_{start} = F_{end}$), we can calculate a conditional probability with this edge as $p(s|F_{start}) = \frac{p(F_{end})}{p(F_{start})}$.

To sequentially estimate the learning difficulty, we start from an empty set and update the graph with the skills one-by-one. Specifically, we regard nodes in $\mathcal{F}^t$ as activated and group their out-edges by the incremented skills. When adding a skill, only the out-edges in the corresponding group contributes to its learning difficulty. Meanwhile, the end nodes of these edges are the newly found frequent subsets. We activate these nodes to obtain $\mathcal{F}^{t+1}$. We describe the details of this process in Algorithm 2. Notably, all the edges are activated at most once, so the time cost is approximately the number of the total activated edges, which is $O(\sum_{F \in \mathcal{F}^*} |F|)$, where $\mathcal{F}^*$ denotes the frequent subsets of the complete skill set.

4 SKILL RECOMMENDER

In this section, we introduce our skill recommender in detail. Specifically, we first give a multi-objective RL formulation of the skill recommendation problem. Then, we describe the structure of SRDQN. Finally, we describe our candidate selection method.

4.1 Multi-Objective RL Formulation

At the *t-th* step of skill selection, we define *state* as the talent's current skill set O_t and *action* as learning a new skill s_t. The *state*

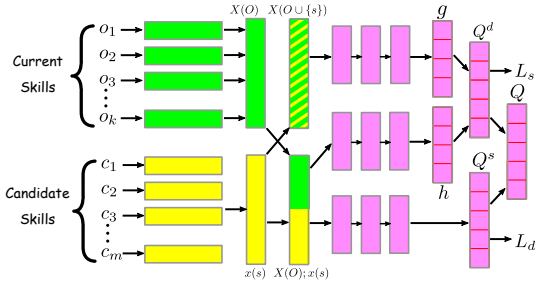


Figure 3: Overall Structure of multi-task Deep Q-Network.

dynamic can be written as $O_{t+1} = trans(s_t, O_t) = O_t \cup \{s_t\}$. Taking an action brings two rewards, written as $r_s(s_t|O_t)$ and $r_d(s_t|O_t)$, where $r_s(s_t|O_t) = f_s(O_t \cup \{s_t\}) - f_s(O_t)$ denotes the salary increase and $r_d(s_t|O_t) = f_d(s_t, O_t)$ denotes the learning difficulty. Supposing that at each step, the talent has a probability $\gamma (0 < \gamma < 1)$ to learn the next skill and otherwise stop, the expected total future salary increase and difficulty of O_t can be written as

$$G_s^t = r_s(s_t|O_t) + \sum_{k=1}^{c} \gamma^k r_s(s_{t+k}|O_{t+k}),$$

$$G_d^t = r_d(s_t|O_t) + \sum_{k=1}^{c} \gamma^k r_d(s_{t+k}|O_{t+k}),$$

where c is the horizon. Under the RL framework, if we regard γ as the discount factor, G_s^t and G_d^t are the long-term return of salary and difficulty, respectively.

Maximizing G_s^t and minimizing G_d^t are two objects of our recommender. To balance them with a unified policy, we define the total long-term return as $G^t = G_d^t - \alpha G_s^t$, where $\alpha \in \mathbb{R}$ is referred as the difficulty importance factor. It is notable that

$$G^t = r_s(s_t|O_t) - \alpha r_d(s_t|O_t) + \sum_{k=1}^{c} \gamma^k (r_s(s_{t+k}|O_{t+k}) - \alpha r_d(s_{t+k}|O_{t+k})).$$

We define $r(s_t|O_t) = r_s(s_t|O_t) - \alpha r_d(s_t|O_t)$ as the *reward*.

In RL, the long-term return of taking action s_t under state O_t following policy π is usually estimated with the Q-value $Q_\pi(s_t|O_t)$. As a common practice, we set a deterministic policy of choosing the skill with the highest Q-value, i.e., $\pi(O_t) = \arg\max_s Q_\pi(s|O_t)$. Then, the Bellman Equation [18] can be written as $Q_\pi(s_t|O_t) = r(s_t|O_t) + \gamma \max_{s'} Q_\pi(s'|O_{t+1})$. Furthermore, we can formulate the salary Q-value $Q_\pi^s(s_t|O_t)$ and difficulty Q-value $Q_\pi^d(s_t|O_t)$ as

$$Q_\pi^s(s_t|O_t) = r_s(s_t|O_t) + \gamma Q_\pi^s(s_{t+1}|O_{t+1}),$$

$$Q_\pi^d(s_t|O_t) = r_d(s_t|O_t) + \gamma Q_\pi^d(s_{t+1}|O_{t+1}),$$

where $s_{t+1} = \arg\max_{s'} Q(s'|O_{t+1})$. Notably, Q^s and Q^d has explicit physical meanings, which correspond to G_s and G_d respectively. Besides, they can be combined for obtaining the unified Q-value function, written as $Q_\pi(s_t|O_t) = Q_\pi^s(s_t|O_t) - \alpha Q_\pi^d(s_t|O_t)$.

4.2 Skill Recommendation Deep Q-Network

In this part, we describe SRDQN in detail, whose structure is shown in Figure 3. Specifically, we first introduce the overall multi-task structure of SRDQN. Then, we introduce our specially designed tower layer for modeling the Q-value, which takes advantage of the deterministic state dynamic of our problem to improve the model performance. Finally, we describe the training process.

4.2.1 Multi-Task Q-value Estimation. Following the above inductions, SRDQN estimates Q^s and Q^d with a specially designed multi-task structure, which can provide the users with explainable long-term utility information. This brings the system with higher interpretability. Then, it combines the two Q-values to obtain a unified recommendation policy. Specifically, we suppose that the two tasks share the shallow action (skill) and state (skill set) representations. The bottom layers trains an embedding vector for each skill, denoted by $x(s) \in \mathbb{R}^{d_e}$. Resulting from the monotonicity of difficulty and salary estimation, the skill set intuitively has a cumulative effect. Specifically, newly-added skills should not weaken the effect of the other skills. Therefore, we embed the state by summing up the skill embeddings, denoted by $X(O) = \sum_{o \in O} x(o)$. Then, for each candidate skill s, we combine $x(s)$ with $X(O)$ and use separate tower layers to model the two Q-values. The detailed structure of the tower layers will be described in Section 4.2.2.

We iteratively train the model parameters with the bootstrap method [18], where the training labels are constructed based on the estimation of the previous model. Specifically, the loss functions of the salary tower and difficulty tower can be formulated as

$$L_s = \mathop{\mathbb{E}}_{O, s \sim \mu(\cdot, O; \Phi^-)} [(r_s(s|O) + \gamma \max_{s' \in C(O')} \hat{Q}^s(s', O'; \Phi^-) - \hat{Q}^s(s, O; \Phi))^2],$$

$$L_d = \mathop{\mathbb{E}}_{O, s \sim \mu(\cdot, O; \Phi^-)} [(r_d(s|O) + \gamma \max_{s' \in C(O')} \hat{Q}^d(s', O'; \Phi^-) - \hat{Q}^d(s, O; \Phi))^2],$$

where $O' = O \cup \{s\}$, Φ and Φ^- are the current and previous model parameters, $\hat{Q}^s$ and $\hat{Q}^d$ are the estimated salary and difficulty Q-value functions, $\mu(\cdot, O'; \Phi^-)$ stores the chosen probability of each skill given state O' under the current policy. We combine these two parts into the model's loss function as $L = L_s + \lambda L_d$, where $\lambda \in \mathbb{R}$ is a super-parameter balancing the importance of these two losses.

4.2.2 Tower Layers for Deterministic State Dynamic. The large state-action space of the skill recommendation problem brings difficulties for fitting the Q-value function. To raise the model performance, we design a special dual structure. Specifically, it is notable that the *state dynamic* of our proposed skill recommendation problem is deterministic. This means for a given state-action pair, the next state is not fixed. Moreover, it is common that two different states transit to the same state. For example, both state $O_0 = \{s_1, s_2\}$ and $O_1 = \{s_1, s_3\}$ transit to $O' = \{s_1, s_2, s_3\}$ by learning skill s_3 and s_2 respectively. In this case, supposing s^* is the best action under state O', we have $Q(s_3|O_0) = r(s_3|O_0) + \gamma Q(s^*|O')$ and $Q(s_2|O_1) = r(s_2|O_1) + \gamma Q(s^*|O')$. For ease of presentation, here we omit the subscript distinguishing salary and difficulty. Notably, the two equations share the second term, which is a function of the next state. Therefore, we model $Q(s|O)$ as $h(s, O; \Phi_h) + g(O \cup \{s\}; \Phi_g)$, where Φ_h and Φ_g are the model parameters. In this way, SRDQN can raise model performance by seizing the relation between Q-value of different states. Specifically, we concatenate $X(O)$ and $x(s)$ and use multi-layer perceptron (MLP) to model $h(s, O; \Phi_h)$. Observing that $X(O \cup \{s\}) = \sum_{o \in O \cup \{s\}} x(o) = X(O) + x(s)$, we sum up $X(O)$ and $x(s)$ to obtain the embedding of the next state. Then, we use MLP to model $g(O \cup \{s\}; \Phi_g)$. Specially, since difficulty is measured with conditional probability, while salary is measured by directly matching the job postings. This causes the measured difficulty smoother

than salary. The smoothness largely reduces the long-term uncertainty of difficulty and makes it more easily modeled. Therefore, we only apply this structure to model Q^s and simply use the concatenated vector to model Q^d. It should be noticed that our formulation is different from the dueling deep Q-network structure [26], which also contains two parts of modeling. Specifically, our motivation comes from the deterministic state dynamic and the widely shared next-state of the problem, so the second part of our formulation is a function of the future state instead of the current state.

4.2.3 Training Process. As we have described before, SRDQN recommends skills for the given skill set and obtain rewards for model training. During the training process, we use ϵ-greedy strategy to sample actions from the candidate skills. Specifically, SRDQN recommends the candidate skill that has the highest Q-value estimation. But with a probability of ϵ, it samples a skill according to the uniform distribution. Formally, the choosing probability $\mu(s, O)$ of skill s given state O can be formulated as

$$\mu(s, O; \Phi^-) = \begin{cases} \frac{1-\epsilon}{|C(O)|} & s \neq \arg\max_{s' \in C(O)} Q(s', O; \Phi^-) \\ \epsilon + \frac{1-\epsilon}{|C(O)|} & s = \arg\max_{s' \in C(O)} Q(s', O; \Phi^-) \\ 0 & s \notin C(O), \end{cases}$$

where $C(O)$ denotes the candidate pool of skill set O. The detailed information of candidate selection can be found in Section 4.3.

The pseudo-code of the training process is shown in Algorithm 3, where we apply the experience replay strategy [18] to achieve off-policy training. Specifically, we store the recommendation records, denoted by (O, s, r_s, r_d), in a memory cache D and randomly draw a subset, denoted by D^{batch}, for each training iteration. In this way, the loss functions can be written as

$$L = \sum_{(O,s,r_s,r_d) \in D^{batch}} (r_s + \gamma \max_{s' \in C(O')} \hat{Q}^s(s', O'; \Phi^-) - \hat{Q}^s(s, O; \Phi))^2 + \lambda \sum_{(O,s,r_s,r_d) \in D^{batch}} (r_d + \gamma \max_{s' \in C(O')} \hat{Q}^d(s', O'; \Phi^-) - \hat{Q}^d(s, O; \Phi))^2,$$

where O' denotes $O \cup \{s\}$.

4.3 Candidate Selection

Since the action space in this work is large, exploring all the skills brings high time complexity and raises the difficulty of model convergence. Intuitively, skills related to the current skill set are more likely to bring high utility. For one thing, less related skills are less likely to achieve a significant salary increase within limited steps. For the other thing, they have higher learning difficulties. For this consideration, we propose a graph-based candidate skill selection method to filter skills related to the current skill set. Specifically, we construct a skill graph where each node represents a skill. An edge connects two skills if their co-appearance frequency is large enough. We count each skills' neighboring nodes in the current skill set as a rough measurement of the relevance. We regard the N_c skills with the highest relevance as the candidate skills. Then, the model only selects skills from the candidate pool, which not only reduces the time complexity for Q-value estimation but also raises data efficiency during training.

Algorithm 3 Parameter Training for SRDQN.

Input: $\mathcal{I}$: the set of initial skill sets; T: number of steps in an episode; N_{batch}: batch size for training; IT_{update}: update period; IT_{max}: training iterations; τ: learning rate;
Output: Φ^-: parameter of the trained model;

```
Initialize an empty memory set D;
for it = 1 → IT_max do
    O^1 ← draw a skill set from I;
    for t = 1 → T do
        C ← find candidate skills of state O^t;
        Sample s ~ μ(·, O^t; Φ^-) from C;
        D ← D ∪ (s, O^t, f_s(s, O^t), f_d(s, O^t));
        O^{t+1} ← O^t ∪ {s};
    for i = 1 → N_batch do
        (s, O, r_s, r_d) ← draw a training sample from D;
        C ← find candidate skills of state O ∪ {s};
        s* ← arg max_{s'∈C} Q̂(s', O ∪ {s}; Φ^-);
        y_s ← r_s + γ Q̂^s(s*, O ∪ {s}; Φ^-);
        y_d ← r_d + γ Q̂^d(s*, O ∪ {s}; Φ^-);
        dΦ ← dΦ + ∂(y_s − Q̂^s(s, O; Φ))^2/∂Φ + λ ∂(y_d − Q̂^d(s, O; Φ))^2/∂Φ;
    Φ ← Φ + τ dΦ;
    if it mod IT_update == 0 then
        Update target network Φ^- ← Φ;
return Φ^-;
```

5 EXPERIMENTS

5.1 Experimental Setup

In this section, we evaluate the performance of our approach with extensive experiments on the real-world dataset.

5.1.1 Dataset. The descriptions of the real-world job posting dataset can be found in Section 2.1. We conducted experiments on the IT-related jobs since they are representative skilled occupations. We first applied *Jieba* [24] to perform word segmentation on the job descriptions and filtered the n-grams with high appearing frequencies. After manually selecting and merging the words, we got 987 IT-related job skills. To reduce noise, we filtered out the job postings containing less than 10 skills. As a result, we got 805,182 job postings in total.

5.1.2 Frequent Itemset. By regarding each job posting as a combination of skills, we mined frequent skill sets with FPGrowth [13], which is a popular frequent pattern mining algorithm, implemented in Spark.MLLib [16]. Specifically, we empirically regarded a skill set as frequent when its support (i.e., θ_d) is larger than 0.005 in this work. Then, we filtered the sets containing no larger than 10 skills and got 374,616 frequent skill sets in total.

5.1.3 Initial Set Construction. We sampled a set of initial skill sets from the job posting data to train and validate our model. Due to market demand differences, jobs are not evenly distributed in the market. Therefore, to comprehensively train and validate the model, we iteratively sample the sets to balance the job duty distribution. Specifically, at each iteration, we measured the total skill appearance frequency of a job posting J^j as $\sum_{s \in J^j} \#(s)$, where $\#(s)$ denotes the number of the current selected initial sets that contain skill s. Then, we randomly selected 100 samples from the 50,000 job postings that have the lowest frequency empirically. This process was repeated

Table 2: Performance Evaluation for Skill Recommendation.

Type	Method	Salary									Difficulty
		4-step	6-step	8-step	10-step	12-step	14-step	16-step	18-step	20-step	
Random	Uniform	1.87	1.94	2.03	2.11	2.20	2.29	2.39	2.48	2.59	3.97
	Frequency	2.09	2.28	2.49	2.73	3.00	3.30	3.63	3.97	4.39	3.11
Greedy	Salary	7.68	11.17	14.81	18.66	22.53	26.16	29.32	31.92	34.11	1.24
	Difficulty	4.82	6.93	9.38	11.98	14.86	17.97	21.07	23.89	26.28	**0.29**
	Reward	**7.92**	11.77	15.84	19.92	23.84	27.35	30.36	32.90	35.09	0.79
Utility Model	Action DNN	3.67	4.67	5.96	7.50	9.57	12.33	15.07	17.60	20.09	0.42
	Salary DNN	3.47	4.45	5.57	6.70	7.93	9.06	10.22	11.27	12.28	1.90
	Difficulty DNN	3.12	4.08	5.35	7.04	9.20	11.80	14.66	17.71	20.72	0.42
	Reward DNN	3.06	3.92	4.87	6.02	7.50	9.18	10.95	12.69	14.38	1.78
Utility Model + RL exploration	Salary DNN	6.42	9.42	12.65	15.96	19.24	22.37	25.24	27.93	30.33	1.23
	Difficulty DNN	4.72	6.80	9.18	11.78	14.66	17.83	20.99	23.89	26.38	0.32
	Reward DNN	6.86	10.52	14.71	19.16	23.56	27.50	30.82	33.49	35.69	0.64
SRDQN	SQDQN(single)	6.74	11.19	16.97	22.58	27.17	30.85	33.84	36.23	38.22	0.62
	SQDQN(MLP)	6.76	11.11	16.50	22.08	26.78	30.51	33.54	35.88	37.75	0.60
	SRDQN	7.03	**11.93**	**17.95**	**23.57**	**27.69**	**31.16**	**34.07**	**36.33**	**38.24**	0.66

until we got 150,000 skill sets. From these sets, we randomly picked 5,000 test samples and used the rest as training samples.

5.1.4 Configurations. Unless specified, we used a set of default settings in our experiment. Specifically, for the environment, the parameter w, and threshold θ_s are set to be 5 and 0.9 respectively, the salary estimator averages the top-100 job salary. For the recommender, we set the candidate pool size (i.e., N_c) to be 100, the difficulty importance factor (i.e., α) to be 0.1, the discount factor (i.e., γ) to be 0.8, and loss function parameter (i.e., λ) to be 1. We trained the model with RMSProp [38] optimizer. The weights are initialized with glorot normal initializer [11]. All the MLPs in the network were composed of 5 fully connected layers with 256 hidden units, which is the same as the embedding dimension.

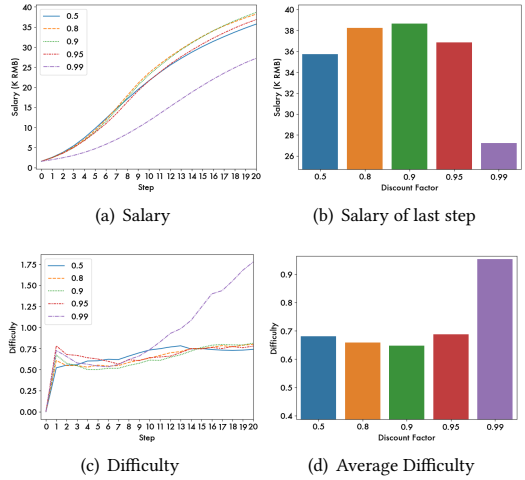
5.1.5 Baseline Methods. Following the real-world skill learning strategies, we constructed several baseline methods, including:

- Randomized selection. We randomly picked from the candidate pool according to 1) uniform distribution and 2) normalized appearing frequency.
- Greedy methods. We recommended the candidate skill which 1) achieves the highest instant salary, 2) has the lowest learning difficulty, 3) has the highest instant reward.
- Model-based methods. We trained a set of Deep Neural Network (DNN) models that take the state as the input and predict the instant utilities of each candidate skill. Specifically, the utilities include the appearing probability ("Action DNN"), instant salary benefit ("Salary DNN"), learning difficulty ("Difficulty DNN"), and reward ("Reward DNN"). We constructed training data from the job postings to train these models and recommend the skill with the highest prediction.
- Model-based methods enhanced with exploration. We utilized the exploration strategy of SRDQN to actively acquire more training samples, and thus strengthen "Salary DNN", "Difficulty DNN", and "Reward DNN". Specifically, we utilized ϵ-greedy to sequentially sample actions and stored the records into the dataset, which is the same process as our RL model. Then we replayed these records to train the models.

5.2 Performance Evaluation

For each sample in the test set, we recommended 20 steps of skills with SRDQN and the baseline methods. In Table 2, we show the averaged difficulty and salary at different steps.

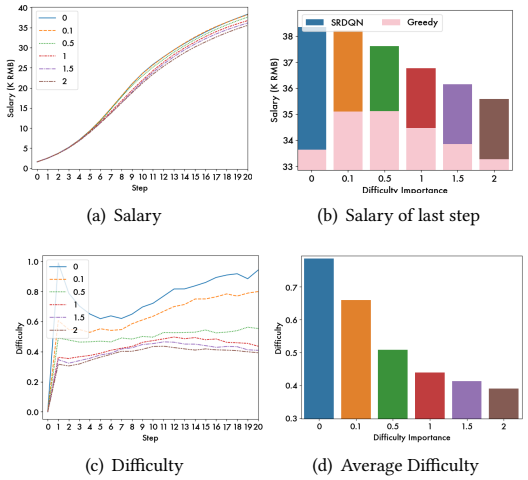
5.2.1 Baseline Performance Analysis. It can be observed that randomized methods perform badly, implying blind learning benefits little for career development. Greedy methods perform well, among which Difficulty Greedy and Salary Greedy assure one-sided high utility but lose the performance on the other perspective. Reward Greedy combines salary and difficulty, which not only has lower difficulty than Salary Greedy but also increases the salary benefit. This phenomenon implies that for skills that bring similar short-term benefits, the easier one is more likely to bring higher long-term benefit. The greedy methods achieve good performance because they accurately calculate the instant utility. However, it should be noticed that they have high time complexity and are sensitive to data noise. For example, if a company posts abnormally highly-paid jobs, these methods will overestimate their required skills' salary benefit without noticing the anomaly. Compared with the greedy methods, the utility models also greedily take actions, but they approximate the utilities with model parameters. Though losing accuracy of value estimation, they are faster and can better resist the noise. However, we can observe from Table 2 that it is difficult



(a) Salary (b) Salary of last step

(c) Difficulty (d) Average Difficulty

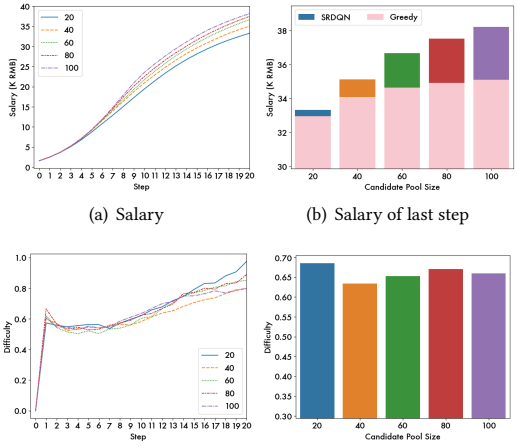
Figure 4: Influence of discount factor γ.



(a) Salary (b) Salary of last step

(c) Difficulty (d) Average Difficulty

Figure 5: Influence of difficulty importance factor α.



(a) Salary (b) Salary of last step

(c) Difficulty (d) Average Difficulty

Figure 6: Influence of candidate pool size N_c.

to train these models with traditional supervised learning methods with limited job posting data.

5.2.2 SRDQN Performance Analysis. There are several observations. First, SRDQN outperforms all the baseline models, proving the effectiveness of our proposed model. Second, by utilizing the exploring strategy of SRDQN, the utility models can be better trained and achieve similar performance with the greedy-based methods. For "Reward DNN" and "Difficulty DNN", the long-term salary is higher than the simple greedy methods because they reduce the data noise. This demonstrates that the active exploration of SRDQN generates more training data, which keeps the model from overfitting and thus raises the model performance. Third, even when adopting the same exploring strategy, SRDQN still outperforms the utility models, which demonstrates the effectiveness of SRDQN in terms of sustainable recommendations. Fourth, greedy-based methods perform better than SRDQN at the first several steps. As we have discussed, their inborn advantages come from the exhaustive requests for the instant reward. But after four steps, SRDQN outperforms all these methods, implying that the long-sighted modeling makes up the accuracy loss of parameter approximation. Fifth, SRDQN generally achieves low learning difficulty. Though there are baseline methods that achieve lower difficulty, they all show a great loss on salary benefit. SRDQN no doubt has a great advantage over them because it assures high salary benefits when keeping low learning difficulty.

5.2.3 Ablation Experiment. We disabled some parts of the network to reveal the effects of the components in SRDQN. Specifically, we first disabled the multi-task structure and let the model to only model the mixed Q-value function, denoted by "SRDQN(single)" in Table 2. The performance decreases a lot at the early steps of recommendations, revealing that the separate Q-value estimation not only raises the model interpretability but also can increase the fitting ability of the model by knowledge sharing between the two tasks. Second, we disable the two-part modeling of the salary Q-value tower and substitute it with a simple MLP structure, denoted by "SRDQN (MLP)". The performance decreases a lot, showing the effectiveness of our specially designed tower layers.

5.3 Parameter Experiment

There are mainly three important parameters that influence the recommendation of SRDQN, which are the discount factor γ, difficulty importance α, and candidate pool size N_c. Here we adjust their value to show their affects. Specifically, for each factor's influence on salary or difficulty, we draw figures to show the overall trend. Since the scale from the first step to the last step may be large, we further draw figures to show the salary performance of the last step and the averaged difficulty.

5.3.1 Discount Factor. In Figure 4, we show the salary and difficulty for models trained with different discount factors γ in the range of $(0.99, 0.95, 0.9, 0.8, 0.5)$, keeping $\alpha = 0.1$ and $N_c = 100$. It can be observed that the trends of salary increase and difficulty are similar for most discount factors. Generally, larger value leads to lower short-term but higher long-term utility. For example, when $\gamma = 0.5$, the model achieves the highest salary and the lowest difficulty at the first 3 to 4 steps but soon gets caught up by the other settings. When γ in $(0.8, 0.9, 0.95)$, the salary increase speeds

up quickly and the learning difficulty decreases in the long-term. These results prove that γ balances the trade-off between short-term and long-term benefits. It is notable that when the value is too large (i.e., $\gamma = 0.99$), the model fails to converge to a proper Q-value estimation. According to Figure 4(b) and Figure 4(d), $\gamma = 0.9$ achieves the best long-term utility. In this paper, we set $\gamma = 0.8$ by default, though it has a similar long-term performance with $\gamma = 0.9$, its short-term performance is better.

5.3.2 Difficulty Importance. In Figure 5, we show the salary and difficulty for models trained with difficulty importance factor α in the range of $(0, 0.1, 0.5, 1, 1.5, 2)$, keeping $\gamma = 0.8$ and $N_c = 100$. It can be observed that α balances the trade-off between salary and difficulty, a larger value significantly decreases the learning difficulty while brings down the salary benefit, especially in the long-term. Specifically, when α is larger than 0.5, the difficulty remains low for the whole recommendation. In contrast, when γ is smaller, the learning difficulty increases as the recommendation gets longer. However, Figure 5(d) implies that laying more importance to difficulty has a negative influence on long-term salary benefit. So in our experiment, we set the default value of α to be 0.1 to assure high salary benefit. Furthermore, in Figure 5(b), we also visualize the salary of the last step achieved by the representative baseline method, Reward Greedy, as a comparison. It can be observed that SRDQN always outperforms the baseline method. Notably, when the difficulty importance factor is set to be 0, the greedy method performs badly while SRDQN still performs well. This implies that difficulty information implicitly enriches the greedy model's ability for long-term decisions. In contrast, since SRDQN has a strong ability to directly estimating the long-term salary increase, difficulty estimation is not necessary for raising long-term salary benefits.

5.3.3 Candidate Pool Size. In Figure 6, we show the salary and difficulty for models trained with candidate pool size N_c in the range of $(20, 40, 60, 80, 100)$, keeping $\gamma = 0.8$ and $\alpha = 0.1$. To give a more explicit understanding, in Figure 6(b), we also visualize the salary of the last step achieved by Reward Greedy as a comparison. In this way, we can reveal how SRDQN explores the candidate set for achieving the long-sighted recommendation. It can be observed that raising candidate pool size affects little on the learning difficulty but has a positive influence on long-term salary benefits. Generally, bigger N_c leads to higher performance. It is notable that increasing candidate pool size affects little on the baseline model performance. The reason is that the baseline models only consider the instant utility and high instant utility. No matter salary or difficulty, the higher utility will be brought by highly relevant skills, which appear at the top of the candidate pool. In contrast, SRDQN can find the skills that are less relevant with the current skill set but potentially bring high long-term benefits. Therefore, a larger candidate pool can bring up long-term performance by enlarging the searching space of SRDQN. This result proves that our SRDQN has a strong ability of long-sighted skill recommendation. By observing Figure 6(b), the increase in performance gets slow when the candidate pool gets larger. Since a large candidate pool will bring down the model efficiency, we set $N_c = 100$ by default.

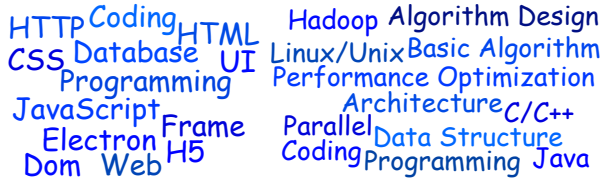
5.4 Case Study

To get more insight into the recommendation process of SRDQN, we did a case study with two skill sets, which are shown in Figure 7. It can be observed that the first skill set is about web development, and the second mainly contains generic programming skills, which may be owned by a CS student. To simplify the analysis, we set $N_c = 50$ in this part.

5.4.1 Candidate Skill Analysis. In Figure 8, we show the candidate skills of each skill set, where the word size represents their relevance with the initial set (defined in Section 4.3). It can be observed that the candidate skills do have strong semantic connections with the skill set, which intuitively can bring high salary benefits with an acceptable learning difficulty. For the first case, most candidate skills are deeply related to the current job duty, among which *"Node.js"*, *"HTTP"*, and *"React.js"* have the highest relevance, leading to low learning difficulty. Besides, there is no doubt that these relevant skills can raise the talent's competency in web development-related jobs. For the second case, since the skill set is more generic and fundamental, its candidate skills are not limited to specific job duties, which include skills about web development, testing, internet services, etc. Indeed, for a CS graduate, their solid fundamental skills can help them quickly adapt to various job positions. It is notable that web development skills still appear a lot. Indeed, since there are many web development job openings in the market, owning these skills can help the talent find a job more easily.

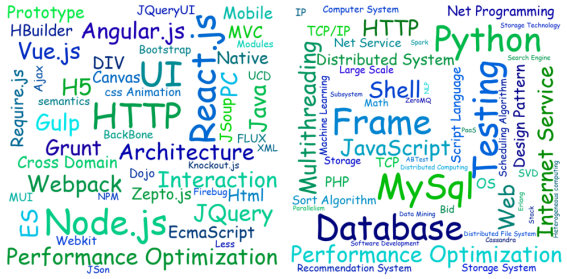
5.4.2 First Step Recommendation. SRDQN makes the recommendation according to the long-term utility estimation of each candidate skill. In Table 3, we list the salary Q-value, difficulty Q-value, instant salary estimation, and instant learning difficulty of the top-10 skills recommended by SRDQN. It can be observed that SRDQN can provide reasonable and long-sighted decision auxiliary. Generally, the recommended skills all have considerable long-term and short-term advantages for. For example, skills like *"Interaction"* and *"Front Performance"* in the first case have high instant benefits. Indeed, instant reward plays an important role in the total reward, i.e., the discounted summation. Some skills have a relatively lower short-term benefit, but SRDQN finds them to have a high long-term benefit, such as *"Math"* in the second case. This is reasonable because many complicated job duties require the talent's math foundation. Besides, though some skills have relatively lower salary benefits, they are recommended if they are easy to be learned, such as *"Spark"* in the second case. Similarly, some relatively difficult skills will be recommended for high salary benefits, such as *"ES"* in the first case. These results show SRDQN to comprehensively balance the multi-perspective benefits instead of providing single-sided recommendations. The talents can refer to these quantifications and choose the skill according to their preference.

5.4.3 Sequential Recommendation Visualization. For each case, we recommended 10 skills with SRDQN. Specifically, at each step, SRDQN chooses the skill with the highest Q-value. In Figure 9, we show the recommended skills and plot the salary and learning difficulty of each step. There are several observations. First, SRDQN can recommend sustainable and realistic skill-learning paths according to the situation. Specifically, for the first case, SRDQN



(a) Case 1 (b) Case 2

Figure 7: Initial sets for case studies.



(a) Case 1 (b) Case 2

Figure 8: Wordcloud of the candidate skills.

suggests improving web development skills since the talent is already familiar with this field. And for the second case, while the talent has multiple choices of career direction, SRDQN suggests learning towards data mining jobs. Considering that data mining jobs are indeed highly-paid in recent years, this result demonstrates the strong ability of SRDQN to seize market demand. Second, both paths have significant salary increases and acceptable difficulty, but they still have some differences. Specifically, for the first case, the learning difficulty is generally lower and the salary increases fast at first several steps. But the speed slows down soon, showing the space of salary increase becomes narrow. This implies that though skill improvement is not difficult, the first talent is also easier to hit the ceiling. In contrast, for the second case, the learning difficulty is higher and the salary increase is slow at first. Indeed, data mining skills are naturally more difficult to learn at the beginning. But after accumulating more skills, the talent can achieve a much higher salary in the long term.

It should be noticed that though we show the path of learning the top recommendation at each step, the talent can choose their own learning path from the set of recommended skills according to their preference in practice, as we have mentioned in Section 5.4.2. This case study shows the ability of SRDQN on giving skill learning simulation, which helps the talent to plan their career. With this simulation, talent will be able to smoothly change their expertise and further change their career direction. Specifically, they can choose the skills related to their targeted job position from the recommended skills, which have a reasonable learning difficulty and benefit their current career. Then SRDQN can simulate the learning path and provide the talent with a direct image of the whole process, which lowers the risk of skill choices.

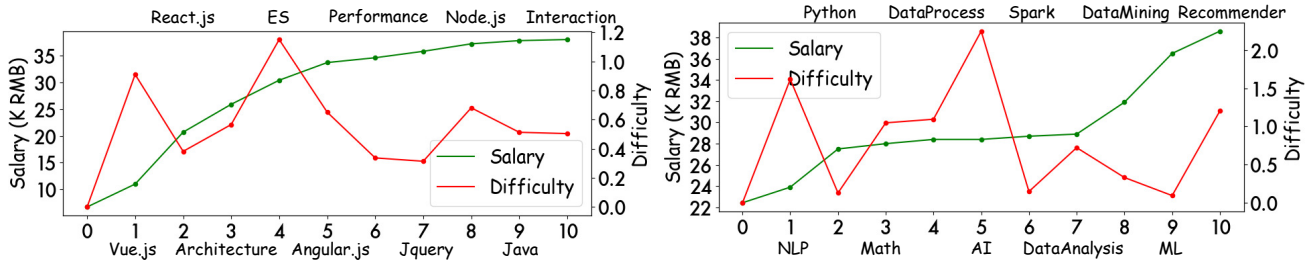
Table 3: Top-10 candidates at the first step.

Case	Candidate	SalQ	DiffQ	Sal	Diff
#1	Vue.js	23.04	8.57	10.86	0.91
	React.js	22.80	8.60	10.25	1.03
	UI	22.30	8.32	13.16	0.72
	Front Performance	22.21	8.11	13.39	0.34
	Iteraction	21.71	8.16	14.05	0.54
	Semantization	21.36	8.83	7.79	1.89
	Node.js	21.13	8.72	9.24	1.28
	ES	20.36	8.84	13.18	1.86
	Architecture	20.20	8.46	11.47	0.57
	Angular.js	20.07	8.48	7.92	1.42
#2	NLP	15.29	8.19	23.90	1.62
	Math*	14.88	8.16	22.62	1.22
	Python	14.70	7.54	25.12	0.23
	Framework	14.58	7.47	26.44	0.19
	Spark	14.18	7.45	22.42	0.24
	Recommending System	14.19	8.36	23.25	1.61
	Testing	13.97	7.71	23.75	0.90
	PHP	13.97	7.91	24.57	1.10
	Database	13.87	7.51	23.49	0.26
	Data Mining	13.79	7.73	22.42	0.64

*Math represents essential math skills for engineers.

5.4.4 Learning Path Comparison. We compared the recommended learning paths of different models for the second case (i.e., fundamental skills), which are shown in Table 4. Generally, the recommended learning path varies for these methods. Specifically, salary-oriented methods (i.e., Salary Greedy and Salary DNN) recommend advanced skills for obtaining high salary benefit. However, they fail to design a proper learning path to master these difficult skills in a sustainable manner. For example, they recommend *"Recommender System"* before *"Data Mining"*, which is unreasonable for *"Recommender System"* largely depends on *"Data Mining"* technology. Especially, Salary Greedy fails to generate a unified learning direction, which recommends a lot of skills related to distributed development before turning to data mining. Second, difficulty-oriented methods (i.e., Difficulty Greedy and Difficulty DNN) tend to start from fundamental skills and gradually recommend more advanced skills. Their learning paths are more sustainable. Indeed, fundamental skills have a lower learning difficulty and can decrease the difficulty of relevant advanced skills. For example, skills like *"Spark"* and *"Python"* make the talent easier to learn to use basic *"Machine Learning"* and *"Data Mining"* tools. However, the learning path is still less focused, leading to lower salary benefit when recommending the same number of skills. Similarly, Action DNN recommends fundamental skills but also brings lower salary benefit. The reward-oriented methods (i.e., Reward Greedy and Reward DNN) decrease the learning difficulty, compared with salary-oriented methods. For example, they replace data mining-related skills with easier skills. However, data mining skills lead to highly-paid jobs in the long-term. Since these methods only consider the short-term utility, they cannot design proper learning paths towards these jobs.

In contrast, SRDQN finds a sustainable learning path that leads to high salary benefit. Specifically, *"NLP"* tools, and *"Python"* are commonly required by jobs in the market and can bring instant benefit. Besides, they can be the base of *"Data Process"*. *"Spark"* helps talent to obtain higher ability on *"Data Analysis"* on big



(a) Case 1 (b) Case 2

Figure 9: Salary and difficulty of the sequential recommendation for case studies.

data, where *"Data Mining"* techniques can be used to for more advanced analysis. Furthermore, *"Machine Learning"* technique enlarges the tasks that the talent can handle. All these skills build the foundation of learning *"Recommender System"*, which is usually related to highly-paid crucial business in Internet companies.

6 RELATED WORKS

This paper's related work can be summarized into two categories, namely Job Skill Recommendation and Deep Reinforcement Learning-based Applications.

6.1 Job Skill Recommendation

With the development of the knowledge economy, choosing the right skill to learn becomes a crucial problem for the talents' career development [15, 25]. Some works have been measuring market demand to provide the talents with skill learning reference [28, 29]. For example, Chootong *et al.* [6] investigated skill demand from job searching websites and recommend in-demand skills to the students. However, due to individual differences, the demanded skills are not always suitable for all the talents. In recent years, more and more efforts have been devoted to recruitment and talent analytic [21, 30, 33], in which personalized job skill recommendation has attracted the wide attention of researchers. Specifically, some works recommend skills according to their relevance to the learner [8, 22]. For example, Dave *et al.* [8] proposed a method to jointly learn the representation of the jobs and skills so that the relevance of new skills to the learner can be measured. Meanwhile, some works recommend skills relevant to jobs that the user may be interested in [2, 19]. For example, Pate *et al.* [19] identified job positions related to the learner's current skill set, and recommend additional skills required for these jobs. Different from the existing works, in this paper, we explicitly modeled the utilities of the skill-learning process and provided sustainable skill recommendations based on deep RL.

6.2 Deep Reinforcement Learning-based Applications

Deep RL is the technology that combines RL principles with deep learning. It learns to properly actions under various states in order to maximize the cumulative reward. In recent years, deep RL has been widely applied in many application scenarios, including games [17, 18, 23], traffic control [27], resources management [14], robotics [12], natural scientific researches [36] and recommendation [35]. For different domains, researchers design domain-specific simulated environments and adjust the deep RL models according to the characteristic of problems. For example, in the field of games, Bellemare *et al.* [3] provides an interface to hundreds of Atari 2600 game environments and Mnih *et al.* [18] uses CNN structure to process the pixel input of Atari games. In the field of traffic signal control, Zhang *et al.* [32] designed CityFlow for city-wide traffic simulation and Zang *et al.* [31] used meta-reinforcement learning to speed up the learning process of new traffic intersections. And for online recommender and advertising systems, user feedbacks are applied as rewards of RL systems [34, 35]. Different from prior arts, in this paper, we design a novel environment to simulate the skill learning process under different situations, and designed a problem-specific deep RL model for skill recommendation.

7 CONCLUSION

In this paper, we introduced a new research problem, namely *sustainable job skill recommendation*, which aims to provide long-sighted, cost-effective, and explainable recommendations for talents who want to learn new job skills. To address this problem, we proposed a data-driven recommendation system based on the technology of deep reinforcement learning. Specifically, we first designed an environment to estimate the utilities of skill learning by mining the massive job advertisement data, including a skill-matching-based salary estimator and a frequent itemset-based learning difficulty estimator. Based on the environment, we specially designed a novel Skill Recommendation Deep Q-Network (SRDQN) with multi-task structure to estimate the long-term skill learning utilities. In particular, SRDQN can recommend job skills in a personalized and cost-efficient manner, which helps talents to gain more salary while pay less learning efforts considering their owned skills. Finally, we conducted extensive experiments on a real-world dataset, which clearly validated the effectiveness and interpretability of our approach. In the future, we will further explore accurate utility estimation to improve the performance and generality of our system, so that it can be applied to more potential applications such as curriculum recommendation.

ACKNOWLEDGMENTS

The research work supported by the National Key Research and Development Program of China (Grant No. 2018YFB1004300), the National Natural Science Foundation of China (Grant No. 61836013, U1836206, U1811461 and 61773361), the Project of Youth Innovation Promotion Association CAS (Grant No. 2017146).

Table 4: The recommended learning paths.

Method	Path
Salary Greedy	Framework→**Distributed System**→**Large Scale**→**GoLang**→**Scala**→Python→Machine Learning→**Recommender System**→**Data Mining**→NLP
Reward Greedy	Framework→Distributed System→Large Scale→GoLang→Shell→Python→PHP→Testing→Net Programming→OS
Difficulty Greedy	Framework→MySql→Database→**Python**→**Spark**→Machine Learning→Shell→**Data Mining**→Performance Optimization→Hbase
Action DNN	Framework→Database→Testing→Web→Python→JavaScript→MySql→HTML→TCP→SQL
Salary DNN	**Recommender System**→Python→Machine Learning→**Data Mining**→NLP→Deep Learning→Large Scale→Math→Scala→Spark
Reward DNN	Framework→Database→Testing→Web→Python→JavaScript→MySql→HTML→TCP→SQL
Difficulty DNN	Framework→Database→MySql→Python→Shell→Spark→Performance Optimization→Machine Learning→Data Mining→JavaScript
SRDQN	**NLP→Python→Math*→Data Process→AI→Spark→Data Analysis→Data Mining→Machine Learning→Recommender System**

*Math represents essential math skills for engineers.

REFERENCES

[1] Timothy T Baldwin and J Kevin Ford. 1988. Transfer of training: A review and directions for future research. *Personnel psychology* 41, 1 (1988), 63–105.

[2] David Baneres and Jordi Conesa. 2017. A life-long learning recommender system to promote employability. *International Journal of Emerging Technologies in Learning (iJET)* 12, 06 (2017), 77–93.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[4] Zahid H Bhat. 2014. Job matching: the key to performance. *International Journal of Research in Organizational Behavior and Human Resource Management* 2, 4 (2014), 257–269.

[5] Ariel Burstein and Jonathan Vogel. 2017. International trade, technology, and the skill premium. *Journal of Political Economy* 125, 5 (2017), 1356–1412.

[6] Chalothon Chootong, Timothy K Shih, Ankhtuya Ochirbat, Worapot Sommool, WKTM Gunarathne, and Carl K Chang. 2019. LCRec: Learning Content Recommendation (Wiki-based Skill Book). *Journal of Internet Technology* 20, 6 (2019), 1753–1766.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[8] Vachik S Dave, Baichuan Zhang, Mohammad Al Hasan, Khalifeh AlJadda, and Mohammed Korayem. 2018. A combined representation learning approach for better job and skill recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1997–2005.

[9] Rafael Dix-Carneiro and Brian K Kovak. 2015. Trade liberalization and the skill premium: A local labor markets approach. *American Economic Review* 105, 5 (2015), 551–57.

[10] Marilyn E Gist, Anna G Bavetta, and Cynthia Kay Stevens. 1990. Transfer training method: Its influence on skill generalization, skill repetition, and performance level. *Personnel psychology* 43, 3 (1990), 501–523.

[11] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[12] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

[13] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. 2008. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*. 107–114.

[14] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 50–56.

[15] Qingxin Meng, Hengshu Zhu, Keli Xiao, Le Zhang, and Hui Xiong. 2019. A hierarchical career-path-aware neural network for job mobility prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 14–24.

[16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[17] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[19] Bharat Patel, Varun Kakuste, and Magdalini Eirinaki. 2017. CaPaR: a career path recommendation framework. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*. IEEE, 23–30.

[20] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Chao Ma, Enhong Chen, and Hui Xiong. 2020. An enhanced neural network approach to person-job fit in talent recruitment. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–33.

[21] Chuan Qin, Hengshu Zhu, Chen Zhu, Tong Xu, Fuzhen Zhuang, Chao Ma, Jingshuai Zhang, and Hui Xiong. 2019. Duerquiz: A personalized question recommender system for intelligent job interview. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2165–2173.

[22] Siddharth Reddy, Igor Labutov, and Thorsten Joachims. 2016. Latent skill embedding for personalized lesson sequence recommendation. *arXiv preprint arXiv:1602.07029* (2016).

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[24] J Sun. 2012. Jieba chinese word segmentation tool. *Accessed: Jun* 25 (2012), 2018.

[25] Chao Wang, Hengshu Zhu, Chen Zhu, Xi Zhang, Enhong Chen, and Hui Xiong. 2020. Personalized Employee Training Course Recommendation with Career Development Awareness. In *Proceedings of The Web Conference 2020*. 1648–1659.

[26] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. 1995–2003.

[27] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 2018. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2496–2505.

[28] Xunxian Wu, Tong Xu, Hengshu Zhu, Le Zhang, Enhong Chen, and Hui Xiong. 2019. Trend-aware tensor factorization for job skill demand analysis. In *IJCAI 2019*.

[29] Tong Xu, Hengshu Zhu, Chen Zhu, Pan Li, and Hui Xiong. 2018. Measuring the popularity of job skills in recruitment market: A multi-criteria approach. In *AAAI 2018*.

[30] Yuyang Ye, Hengshu Zhu, Tong Xu, Fuzhen Zhuang, Runlong Yu, and Hui Xiong. 2019. Identifying high potential talent: A neural network based dynamic social profiling approach. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 718–727.

[31] Xinshi Zang, Huaxiu Yao, Guanjie Zheng, Nan Xu, Kai Xu, and Zhenhui Li. 2020. MetaLight: Value-Based Meta-Reinforcement Learning for Traffic Signal Control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1153–1160.

[32] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. 2019. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The World Wide Web Conference*. 3620–3624.

[33] Le Zhang, Tong Xu, Hengshu Zhu, Chuan Qin, Qingxin Meng, Hui Xiong, and Enhong Chen. 2020. Large-Scale Talent Flow Embedding for Company Competitive Analysis. In *Proceedings of The Web Conference 2020*. 2354–2364.

[34] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1040–1048.

[35] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*. 167–176.

[36] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. 2017. Optimizing chemical reactions with deep reinforcement learning. *ACS central science* 3, 12 (2017), 1337–1344.

[37] Chen Zhu, Hengshu Zhu, Hui Xiong, Chao Ma, Fang Xie, Pengliang Ding, and Pan Li. 2018. Person-job fit: Adapting the right talent for the right job with joint representation learning. *ACM Transactions on Management Information Systems (TMIS)* 9, 3 (2018), 1–17.

[38] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. 2019. A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 11127–11135.