A Future-Oriented Cache Management for Mobile Games

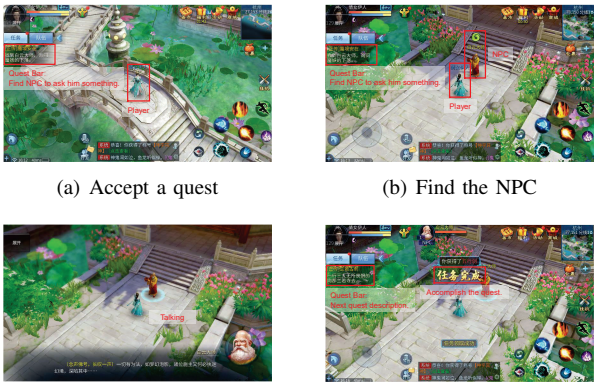
Yue Hu, Meng Wang, Yingfeng Chen (✉), Changjie Fan
Fuxi AI Lab, Netease, Hangzhou, China
{huyue1, wangmeng02, chenyingfeng1, fanchangjie}@corp.netease.com

***Abstract*—As a common but critical module, cache management plays more and more important roles in modern videos games, especially for the games on mobiles whose hardware is weaker than the desktop computers. Current cache management policies typically resort to heuristics designed for the common access patterns, which cannot fully utilize the information in mobile games and may fail on the complex access patterns in it. In this paper, we propose a future-oriented cache management (FOCM) to cope with the cache management problem in mobile games' typical scenarios. FOCM leverages a dual Markov model to bridge the cache management's decision-making process and the behavioural process of the players in games, resulting in modelling the potential access requests in the future more accurately. Based on this model, we propose two cache management methods based on imitation learning and reinforcement learning, with each one suitable for different types of games. The experimental results in both the cache simulation environment and the industrial massively multiplayer online role-playing game (MMORPG) demonstrate the effectiveness of the proposed methods.**

I. Introduction

In recent years, the game industry has developed vigorously [1], [2], in which the mobile games account for nearly half of the market share [3]. Generally speaking, most mobile games run on mobile phones, whose memories are smaller than desktop or laptop computers. Meanwhile, the 3D models used in modern mobile games are more and more complex, which may result in large latency for loading the models when the game scene is initializing or changing, especially on some old phones. A widely used solution is to introduce the cache system and decrease the 3D model loading latency by storing the most popular models in the cache from which the game can access data faster than from the lower-level storage. On the other hand, the game quests system is the footstone of a modern game, which guides players to know the characters and the game world. A typical example in a massive online role-playing game (MMORPG) is shown in Fig. 1. The player is required to accomplish this quest with four steps: accepting the quest, finding the related non-player character (NPC), talking to the NPC and accomplishing the quest. The game quests system is an important sub-module for mobile games nowadays and the number of quests in the game is quite a few, especially in MMORPGs. Thus an obvious reduction of model loading latency in quests scenarios can also improve the player's overall game experience significantly.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE



(a) Accept a quest (b) Find the NPC

(c) Talk to the NPC (d) Accomplish the quest

Fig. 1. A typical example of accomplishing a quest in a mobile MMORPG.

As the capacity of the cache is limited, correctly selecting what *data* to store in the cache is critical for reducing the latency. Prior cache management methods usually rely on manually-engineered heuristics to capture the most common cache access patterns, such as evicting the least frequently used (LFU) or least recently used (LRU) *data*. These data-independent methods are simple and applicable to various kinds of caches. However, because of the lack of adaption to the game scenarios, these methods are commonly not capable of achieving high performance in mobile games. As illustrated in Fig. 1, the players are requested to complete the sub-tasks in a fixed order in quests scenarios, and thus there exist some implicit behaviour patterns in the process of accomplishing a quest. These behaviour patterns imply the *data* (e.g., 3D models) may be used next, which is important for the cache replacement arrangement.

In this paper, we propose a future-oriented cache management (FOCM) framework for the game quests system, which improves the cache scheduling according to the implicit behaviour patterns of the players in games. FOCM bridges the players' behaviours and the cache management's decisions with a dual Markov model, which consists of two Markov decision processes (MDPs). The visible MDP is the decision-making process of the cache management that determines what *data* should be stored in the cache. The hidden MDP is the behavioural process of the players, whose different behaviours leading to various kind of *data* requests. FOCM also includes two cache management methods based on imitation

learning (IL) and reinforcement learning (RL), with each one corresponding to the situations where the transition probability distribution between the hidden MDP and the visible MDP is explicitly accessible or not, respectively. Besides, unlike the previous methods that rely on accessing the histories of the *data* (such as LRU, LFU), our methods try to infer the *data* that are most likely to be used in the future by learning the behaviour patterns of the players directly or indirectly.

The main contributions of this paper are summarized below:

- We propose a novel cache management framework which is optimized for the quests system scenarios in mobile games. Our framework connects the cache management's decisions with the player's behaviours in a dual Markov model and leverages the implicit patterns contained in the player behaviours to increase the cache hit rate;
- We propose two cache management methods based on different learning paradigms, i.e., imitation learning (IL) and reinforcement learning (RL), with the former one for learning the player's behaviour directly and the later one for learning indirectly;
- Experimental results in both the open-source simulated game environment and the real industrial MMORPGs confirm the effectiveness of our modelling method as well as the cache management methods.

II. Background

A. Cache Management

The cache is a faster data storage system, which plays a critical role in improving the systems in various domains, such as content delivery infrastructures, web servers and terminal devices. The cache has a fixed storage capacity C and we assume that every piece of *data* uses the same cache size. The *data* in the cache can be defined as $D^c = \{d_1, d_2, ..., d_c\}$, d_i means the ith *data* and $c \leq C$. The requested *data* can be read from the cache if it exists in the cache, this is called a cache hit. Otherwise, this is a cache miss, the requested *data* need to be read from a slower data storage.

As we all know, the cache is commonly faster than the lower-level storage but the capacity is usually limited, and the goal of cache management is to reduce the overall data reading latency by storing proper data in the faster cache. The main work of the cache management can be divided into three steps: 1) Hit check: comparing the requested *data* with the *data* in the cache to determine whether the cache hits; 2) Caching decision: deciding whether to put the requested *data* into the cache if the cache misses; 3) Location selection: selecting a proper location to put the requested *data* if the cache misses and the requested *data* needs to be stored in the cache.

B. Belady's Policy

Generally speaking, the third step, i.e., the cache replacement, is the most important one in cache management, and it is also the focus of research. In the most ideal situation when the list of all future requested *data* is known, there exists a cache replacement policy named Belady's policy [4]. This policy will compute the reuse distance $dis(d_i)$ of each *data* d_i in the cache, which is defined as the number of the future requests until the next requested *data* is d_i. Then the *data* that has the highest reuse distance will be replaced because it will be used furthest in the future. In Belady's policy, the window size of the future requested *data* list is important for the final performance, and the larger the window size, the better the effect of this policy.

C. Markov Decision Process

A Markov decision processes (MDP) is a discrete-time stochastic control process, generally expressed as a four-tuple (S, A, P, R). S is the state space, A is the action space, $P(s_{t+1} = s^{'}|s_t = s, a_t = a)$ is the probability that action $a \in A$ in state $s \in S$ at time t will lead to state $s^{'} \in S$ at time $t + 1$ and $R(s, s^{'})$ is the reward received from the state s to the state $s^{'}$.

D. Imitation Learning

Imitation learning refers to learning from the examples provided by the instructor, generally the provided example is decision data of human experts like $\{\tau_1, \tau_2, ..., \tau_m\}$. Each decision contains a state and action sequence $\tau_i =< s_1^i, a_1^i, s_2^i, s_2^i, ..., s_n^i >$, and all state-action pairs are extracted to construct a new data set $\mathcal{D} = \{(s_1, a_1), (s_2, a_2), ...\}$. Then making the state as feature and the action as label, we can use supervised learning algorithms to learn a model imitating the human behaviour.

E. Reinforcement Learning

In recent years, reinforcement learning [5]–[9] has been successfully applied to sequential decision-making problems. The reinforcement learning problem is modelled as an MDP, and the goal is learning a policy $\pi : A \times S \rightarrow [0, 1]$ to maximize the expected cumulative reward. The learning process is the agent continuously performs actions in the environment under current policy, and gets a reward from the environment to adjust the policy until it finds a policy that can maximize the cumulative reward. This process allows the agent to keep trial and error in the environment, accumulate experience, and find the optimal policy. Unlike supervised learning algorithms, reinforcement learning does not require large labelled data sets.

III. Methodology

In this section, we first introduce our FOCM framework for solving the cache management problem. Then, we present two specific methods based on the proposed model.

A. Future-Oriented Cache Management

There are three main components in the caching system: the cache, the requested *data* and the cache management method. The cache has a fixed storage capacity C and the *data* in it are denoted as D^c. The request list consists of a series of requested *data*, represented by $Req = \{req_1(d_1), req_2(d_2)...\}$, where $req_i(d_j)$ means the ith request calls for the jth *data*. For brevity, we use req_i represents requested *data* in the later sections. And for each given req_i, the cache management

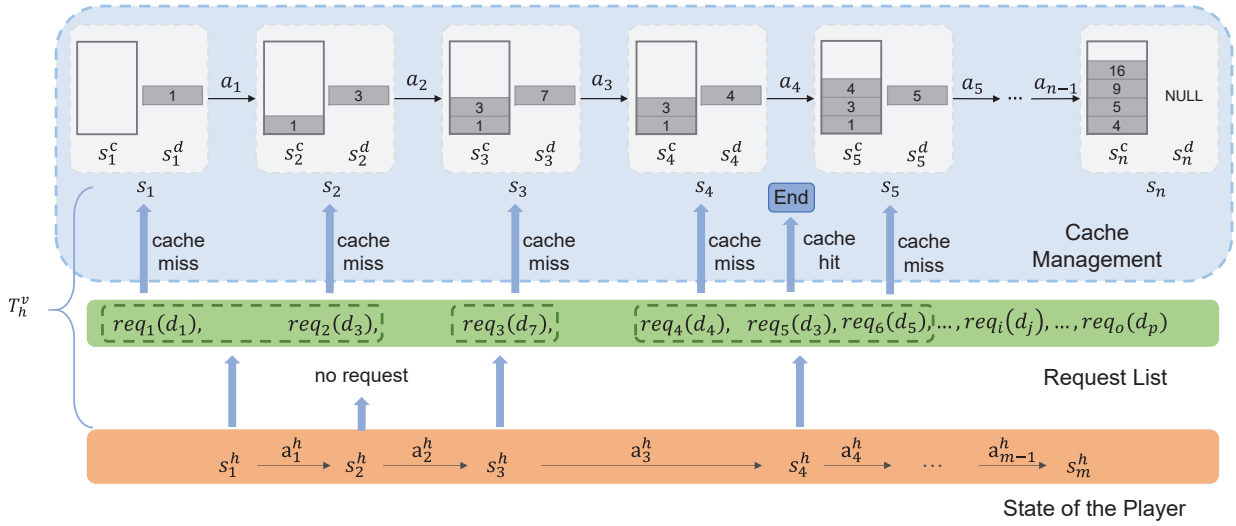


Fig. 2. The framework of the future-oriented cache management (FOCM). The model consists of two MDPs: the visible MDP (cache management decision-making process) and the hidden MDP (requester behavioural process). And the T_h^v represents the transition probability distribution between the hidden MDP and the visible MDP.

method needs to process it following three steps introduced in Section II-A.

For the caching system illustrated above, we propose a future-oriented cache management framework dubbed FOCM to cope with the cache management problem in typical scenarios of mobile games. The model allows for the behaviours of the players because the req_i is closely bound up with the player's behaviour in the game world. The framework of FOCM is illustrated in Fig. 2. Our model leverage a duel Markov model to bridge the decision-making process of the cache management and the behavioural process of the players in games.

The hidden MDP is the behavioural process of the players, i.e., (S_h, A_h, R_h, P_h), $s_i^h \in S_h$ represents the state of the player at timestep t_i, $a_i^h \in A_h$ is the behaviour of the player and P_h is the transition probability. The reward R_h is 1 when the player accomplishes the quest and 0 otherwise. The cache does not contain the player behaviour information explicitly thus the behavioural process of the player is defined as a hidden MDP.

The visible MDP is the process that the cache management operates the cache according to the current status of the cache and the requested *data*, that is, (S, A, P, R). Here, $s_i \in S = [s_i^c, s_i^d]$ represents the combination of s_i^c and s_i^d, with each one corresponding to the cached *data* and the requested *data*, respectively. The action $a_i \in A$ is the operation of the caching system, such as replacing *data* from the cache or putting *data* into the cache. P and R are the transition probability and the reward for the visible MDP respectively, with the latter one related to the hit rate.

To understand our framework more easily, we first assume that the transition probability T_h^v between the hidden MDP and the visible MDP is deterministic. The player and the caching system are linked together during the process of *data* requesting and *data* reading, which is illustrated in Fig. 2. In the game, the player will call for *data* in different states such as encountering monsters or skill special effects. The req_i generated based on the state of the player as follows:

$$\{req_1, req_2, ..., req_o\} = f(s_i^h) \tag{1}$$

where $f(\cdot)$ is the function to calculate the *data* need to be requested in player's visual domain and $o \geq 0$. For example, the player generate requested *data* $(req_1(d_1), req_2(d_2))$ in s_1^h and no request in s_2^h as shown in Fig. 2. The s_i^d is the req_i which occurs cache misses:

$$s_i^d = h(req_i, s_i^c) \tag{2}$$

$$h = \begin{cases} h_1, & req_i \notin D^c, \\ \varnothing, & req_i \in D^c. \end{cases} \quad D^c = k(s_i^c) \tag{3}$$

$$h_1(req_i) = req_i \tag{4}$$

where $k(\cdot)$ is the map from the state of the cached *data* to the D^c, $h(\cdot)$ is the function for hitting check and there will be no map from req_i to s_i^d when the cache hits. For example, when $req_5(d_3)$ is coming, the d_3 can be found in the cache, so the cache hit occurs and there is no corresponding s_i^d in this situation as shown in Fig. 2. The s_i^c is the data storage status in the cache, which depends on the s_{i-1} and a_{i-1}:

$$s_i^c = g(s_{i-1}, a_{i-1}) \tag{5}$$

where $g(\cdot)$ means doing the action a_{i-1} on the state s_{i-1}, the action is putting the requested *data* in the cache or not. The process will be clearer in conjunction with Fig. 2. d_1, d_3, d_7, d_4 are all cache misses, so they have the corresponding state s_1, s_2, s_3, s_4. And a_1, a_2, a_4 denote actions of putting d_1, d_3 and d_4 into the cache. Besides, if the cache management method thinks that the requested *data* will not be required in

the future, it will not put this *data* into the cache even when a cache miss happens. The action a_3 on *data* d_7 is an example of this situation. The relation between the hidden MDP and visible MDP as follows:

$$\begin{aligned} s_i &= [s_i^c, s_i^d] \\ &= [g(s_{i-1}, a_{i-1}), h(req_i, s_i^c)] \\ &= [g(s_{i-1}, a_{i-1}), h(f(s_i^h), g(s_{i-1}, a_{i-1}))] \end{aligned} \tag{6}$$

And if the transition probability is not deterministic, the $P(s_i|s_{i-1}, a_{i-1})$ is defined as follows:

$$\begin{aligned} P(s_i|s_{i-1}, a_{i-1}) &= P(s_i^c|s_{i-1}, a_{i-1}) \cdot P(s_i^d|s_{i-1}, a_{i-1}) \\ &= P(s_i^c|s_{i-1}, a_{i-1}) \cdot P(s_i^d) \\ &= P(s_i^c|s_{i-1}, a_{i-1}) \cdot P(s_i^d|s_i^h, s_i^c) \end{aligned} \tag{7}$$

where $P(s_i^d|s_i^h, s_i^c)$ is the transition probability between the hidden MDP and visible MDP, we defined it as T_h^v.

Good cache management can perform proper operations on the cache, which can improve the overall hit rate, reduce the frequency of data replacement in the cache, and speed up the reading speed. Based on the FOCM framework, the cache management problem will be transformed to finding an optimal action a_i at the state s_i. And in MMORPGs, the current destination of the player is closely tied to his quests, and thus many players' behaviour patterns will be similar. Our FOCM framework takes the player's behaviours into account during the modelling process, which can help the cache management to make decisions by learning the behaviour patterns of the players. To be concrete, FOCM includes two cache management methods based on the behaviour of the player, which are imitation learning based (IL-based) method and reinforcement learning based (RL-based) method.

B. Imitation Learning Based Method

The imitation learning based (IL-based) method predicts the future behaviours of the player first by imitation learning from a data set including a large number of the players' historical behaviours and then inferring the list of the future requested *data* when the T_h^v is explicitly accessible. Finally, the Belady's policy can be used to make decisions according to the future requested *data* list and the current cache status.

The IL-based method imitates the player's behaviours to get a future requested *data* list indirectly. The future requested *data* list can also be got by directly predicting the future requested *data* based on a great deal of historical requested *data*. However, in complex systems such as mobile games, the compounding errors [10] caused by predicting the requested *data* list are much more serious than predicting the player's future behaviours, which is shown in the experiments in Section IV-B. So we design our IL-based method upon the behaviours of the player to get the future requested *data* more accurately.

We get many players' behaviour list $\{\tau_1, \tau_2, ..., \tau_n\}$, where τ_i means a player's behaviour list. Every behaviour list includes a sequence of states and actions, that is, $\tau_i =< s_1^{hi}, a_1^{hi}, s_2^{hi}, a_2^{hi}, ..., s_m^{hi} >$. We extract all state-action pairs to construct a new data set $\{(s_1^h, a_1^h), (s_2^h, a_2^h), (s_3^h, a_3^h), ...\}$. Then we can imitate the player behaviours as follows:

$$a_i^h = F_{player}(s_i^h) \tag{8}$$

where $F_{player}(\cdot)$ is a non-linear function for predicting the behaviour of the player a_i^h based on the current state s_i^h in the game. The $F_{player}(\cdot)$ is usually modelled with neural networks. Then, we use the predicted behaviour to get the next state s_{i+1}^h according to the transition probability $P_h(s_{i+1}^h|s_i^h, a_i^h)$ in the player behaviour MDP (hidden MDP), and the next action a_{i+1}^h can be obtained through inputting the s_{i+1}^h into $F_{player}(\cdot)$. The subsequent states and actions can be obtained in the same manner and finally we can get the whole behaviour track of the player. At last, we can infer the future requested *data* list according to the T_h^v.

After obtaining the future requested *data* list, we can use the planning method to operate the current cache. We use $D^c = (d_1^c, d_2^c, ..., d_i^c, ..., d_c^c)$ to denote the *data* in the cache, $d_{current}$ denotes the current requested *data* and $D^{fr} = (d_1^r, d_2^r, ..., d_j^r, ..., d_m^r)$ denotes the future requested *data* by prediction. There are two cases in the planning method. On the one hand, if $d_{current}$ not in D^{fr}, $d_{current}$ should not be put into the cache, because the *data* will not be requested in the future. On the other hand, we use Belady's policy to find the *data* to be replaced in the cache. Because our window size of the predicted requested *data* list is not infinite, so not every *data* in the cache can compute the reuse distance $dis(d)$. In this situation, the reuse distance is defined as $+\infty$. Based on $dis(d)$, the *data* to be replaced in the cache can be got as follows:

$$d_{replace} = \underset{d \in D_{current} \cup D^c}{arg\ max}\ dis(d) \tag{9}$$

where $d_{replace}$ means the *data* need to be replaced. If $d_{replace} = d_{current}$, $d_{current}$ should not be put into the cache. Otherwise, $d_{replace}$ in the cache will be replaced by $d_{current}$.

C. Reinforcement Learning Based Method

As the decision-making process of the cache management is a sequential decision-making problem, it is appropriate to use the reinforcement learning based (RL-based) method to find the optimal policy to manage the cache.

Using reinforcement learning, we take the cache management module as an agent, which learns to perform different operations on the cache in different states. Formally the cache management process is defined by $G = \langle S, A, P, R \rangle$, where S denotes the state space, A denotes the action space, P denotes the state transition function and R denotes the reward function. The state $s \in S$ including the information of the player, the requested *data* and the cache status. The action space is $A = \{0, 1, 2, ..., C\}$, where C is the cache capacity, and the size of the action space is $C + 1$. When the selected action is 0, it means the requested *data* should not be put into the cache. When action $j \in \{1, 2, ..., C\}$ is selected, it means the requested *data* should be put into the cache in the jth

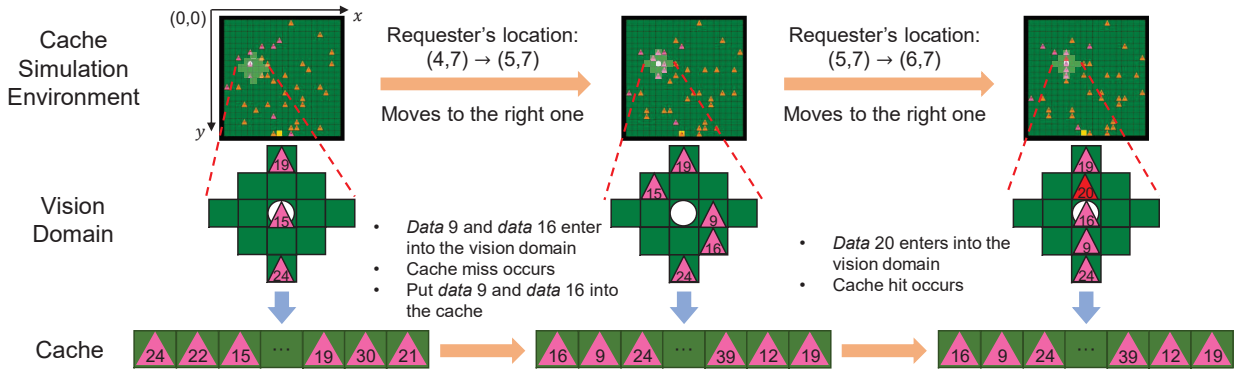


Fig. 3. Illustration of the cache simulation environment. The white circle represents the requester (i.e., player), the yellow square represents the current destination and the triangle represents the *data* and the pink triangle represents the *data* in the cache. We show the process of *data* requesting and loading with the FIFO strategy.

location if this location is blank, or replace the *data* in the jth location if this location is not empty. We define the reward function as follows:

$$R = R_{immediate} + R_{final} \tag{10}$$

$$R_{immediate} = \begin{cases} 1, & hit, \\ 0, & otherwise. \end{cases} \tag{11}$$

$$R_{final} = \alpha \times hitrate \tag{12}$$

where $R_{immediate}$ is the immediate reward, R_{final} is the final reward and α is a hyper-parameter. The immediate reward will take effect at every step and will be 1 if the cache hits and 0 otherwise. The final reward will be activated only at the end of one episode and its value is tied to the episode overall hit rate. The policy π is trained to achieve a higher hit rate. There are many deep reinforcement learning methods that can be applied to this problem, such as DQN [9], DDPG [11], PPO [12], IMPALA [13].

The RL-based method will get the optimal policy after a large number of trial-and-error. And as it does not need the T_h^v to infer the list of the future requested *data* explicitly, it can be applied in the games regardless of whether T_h^v is accessible.

IV. Experiments

In this section, we conduct experiments in a cache simulation environment and a very popular commercial mobile MMORPG in China to evaluate the performance of our method. Firstly, we introduce the environments used in our experiments. Then, we design an experiment to compare the compounding errors of the sequential predictions upon the requested *data* sequences and the player behaviour sequences, respectively. At last, we compare the performance of our methods with the traditional cache management approaches such as LRU and FIFO in the two environments.

A. Experimental Environment

In this section, we introduce our cache simulation environment and the real mobile MMORPG.

1) Cache Simulation Environment: The cache simulation environment simulates the process of the player calling for the *data* and the operations of the cache management. The cache simulation environment includes the player, cache and *data*. As shown in Fig. 3, the white circle represents the requester (i.e., player), the yellow square represents the current destination, and the triangle represents the *data*. The player moves towards the destination along a certain route, which simulates the situations where the player finds a series of NPCs in a specific quest in the real games. When the *data* appears in the player's field of view, it will need to be loaded from the cache if it is in the cache. Otherwise, the cache management needs to decide whether to put the *data* in the cache or not.

The basic settings of this simulation environment are as follows: 1) Each *data* has an id and goes through a series of destination points $DP = \{dp_1, dp_2, dp_3...\}$ in the grid. The subsequent destination dp_{i+1} after arriving at current destination dp_i is sampled from a probability distribution $\mathbb{P}(dp_i)$, which is formalized as follows:

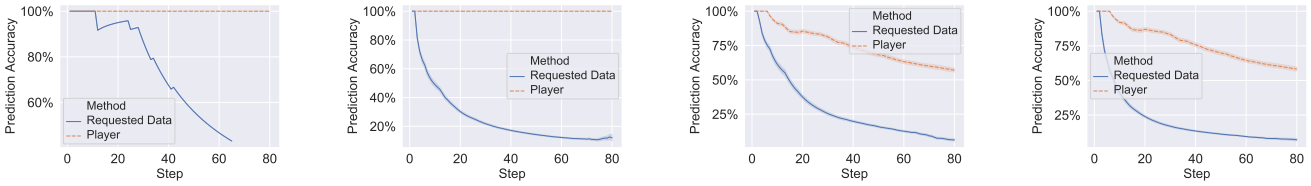
$$\mathbb{P}(dp_{i+1}) = [p(dp_i, dp_1), p(dp_i, dp_2), p(dp_i, dp_3)...] \tag{13}$$

where $p(dp_i, dp_j)$ means the probability to choose dp_j as the next destination after arriving at dp_i. There are two movement modes: stochastic and fixed, and in the fixed mode $\mathbb{P}(dp_i)$ will be a one-hot vector indicating a deterministic dp_{i+1} for each dp_i. Based on this definition, we can represent the overall destination transition process with a matrix P_{dp} as follows:

$$P_{dp} = [\mathbb{P}(dp_1)^T, \mathbb{P}(dp_2)^T, \mathbb{P}(dp_3)^T ...] \tag{14}$$

2) The movement mode of the player is similar to the *data*, which can also be divide into the fixed situation and the stochastic situation. 3) The player will have a vision domain during the movement and the *data* that enter this vision domain need to be loaded. The vision domain of the player is calculated through the Manhattan distance.

Data request will occur during the movement of the player. Assume that the cache capacity is 10, the Manhattan size of the vision domain is 2, and the cache management strategy is FIFO. The process is shown in Fig. 3, where the pink triangle



(a) Fixed player and fixed *data* mode

(b) Fixed player and stochastic *data* mode

(c) Stochastic player and fixed *data* mode

(d) Stochastic player and stochastic *data* mode

Fig. 4. The cumulative prediction accuracy of the method of predicting the requested *data* and the method of predicting the behaviour of player in four different movement modes.

represents the *data* in the cache, and the status of the cache is shown at the bottom of Fig. 3. The number in the cache is the id of the *data* and the order of *data* entering the cache is from right to left.

2) A Real Mobile MMORPG: The real environment is a popular mobile MMORPG in China, which has many daily active users. As shown in Fig. 1, the screen of the mobile phone is the vision domain of the player. During the player's movement in the game, *data* on the screen will be called for. The requested *data* include other players, pets, NPCs or skill effects. The number of the total requested *data* in the game is 55000, and the behaviour of one player will affect the requested *data* of the other players. For example, player A enters the vision domain of player B, then player B needs to call for *data* about player A. We will carry out the experiment during the player accomplishing a quest. As shown in Fig. 1, when the player accepts a quest, the player will find some NPCs orderly to accomplish the quest. The cache management problem in the MMORPG is complicated and difficult.

B. Compounding Error

In this section, we design an experiment to compare the compounding errors of the sequential predictions upon the requested *data* sequences and the player behaviour sequences, respectively.

1) Experiment Settings: In the cache simulation environment, since both *data* and player have two movement modes, there are four modes in total. We will compute the compounding error of two methods in these four modes. And we compute the cumulative prediction accuracy of requested *data* and behaviour as evaluation criteria, the lower accuracy means the larger compounding error. The experimental environment is based on a 20×20 grid with 40 dynamic *data*. The cache capacity is 10 and the size of the vision domain in Manhattan distance is set to 2.

The method of predicting the future requested *data* list is modeled as follows:

$$req_{i+1} = F_{request}(req_i, req_{i-1}, ..., req_{i-j}) \tag{15}$$

where $F_{request}(\cdot)$ is a non-linear function for predicting the upcoming requested *data* based on the recently $j+1$ requested *data* that have been called for and req_i means the ith requested *data*. In our experiment, the $F_{requet}(\cdot)$ is modeled with deep neural network (DNN). The input of the $F_{request}$ is the recently requested 10 *data*, the output is the probability distribution on the *data* space and we choose the *data* with the highest probability as the next requested *data*. The loss of the method is cross-entropy.

The method of predicting the behaviour of player is introduced in Section III-B. The $F_{player}(s_i^h)$ is also modelled by a DNN, and the input is the current state of the requester, including current coordinates and starting point. The output is the probability distribution on the action space and we choose the action with the highest probability as the next behaviour of the requester when we use the model. The loss function and the training process are the same as the predictive model of the requested *data* list.

2) Results and Analysis: As mentioned above, there are four modes in our simulated environment. In each mode, we perform 1000 tests for these two methods respectively. The results are shown in Fig. 4 and each point represents the average cumulative prediction accuracy up to the corresponding step. The yellow dashed line and the blue line represents the results of the behaviour's prediction and the requested *data*'s prediction, respectively. The abscissa represents the number of subsequent steps predicted, we intercept to 80 steps because the number of the player's behaviours and the number of requests are about 80 until the player finishes one episode. And the yellow dashed line in Fig. 4(a) and Fig. 4(b) is the same because the player's behaviours in these two modes are fixed. The situation in Fig. 4(c) and Fig. 4(d) is similar because of the same reason.

In Fig. 4, the four cumulative prediction accuracies of the behaviour's prediction are all higher than the requested *data*'s prediction, which means the compounding error of the former one is tinier than the latter one. The behaviour's prediction is only related to the movement of the player, but the requested *data*'s prediction is related to the movement of the player and the dynamic *data*, so it is relatively difficult to predict the future requested *data* in the requested *data*'s prediction case. In the sequential prediction situation, the subsequent predictions are based on the previous predictions, and thus the errors in the middle predictions will affect subsequent predictions, which raises the difficulty of making an accurate prediction.

C. Performance In the Cache Simulation Environment

1) Experiment Settings: We verify the performance of different methods in the cache simulation environment at four modes. The basic settings of the cache simulation environment

TABLE I
THE AVERAGE HIT RATE OF DIFFERENT METHODS ON FOUR KINDS OF MOVEMENT MODES IN THE CACHE SIMULATED ENVIRONMENT.

Movement Mode		Method					
Player	*Data*	Random	FIFO	LRU	IL-Baed	RL-Based	Belady's Policy
Fixed	Fixed	0.1639	0.1818	0.1667	**0.4091**	**0.4091**	0.4091
Fixed	Stochastic	0.1455	0.1540	0.1451	0.2547	**0.2853**	0.4016
Stochastic	Fixed	0.1379	0.1416	0.1305	**0.2877**	0.2658	0.3937
Stochastic	Stochastic	0.1382	0.1424	0.1350	0.1981	**0.2421**	0.3917

TABLE II
THE AVERAGE HIT RATE OF DIFFERENT METHODS IN THE MMORPG GAME. THE GAME'S BUILT-IN METHOD IS BASED ON THE RULE AND LRU.

Method	Game's Built-In Method	RL-Based Method	Belady's Policy
Average Hit Rate	0.5096	0.5929	0.7221

are the same as the experiment of compounding error. We use the average hit rate as the evaluation criterion of the method performance. We carried out LRU, FIFO, random policy, IL-based method and RL-based method to guide the cache management to manage the cache. At the same time, the optimal solution by Belady's policy can be calculated with all subsequent requested *data* lists known, that is, an upper limit of the average hit rate.

The detailed parameters and model design of the IL-based method are the same as the experiment of compounding error. The RL-based method takes the cached *data*, the requested *data*, the state of the requester, and the route information as the state, the operations of the cache management as the action. The reward function is defined as Equation 10, where α is 500. And we use IMPALA [13] to train the parameters of the network.

2) Results and Analysis: We perform 10,000 tests for each method in the same environment and calculate the average hit rate. The results are summarized in Table I and it shows that our two methods are better than other baselines in all four modes. In the easiest mode where the movements of the player and *data* are fixed, our two methods get to the upper limit whereas the three baselines cannot even reach half of the upper limit. In other more difficult modes, our two methods cannot get the upper limit due to the stochasticity, but compared with the three baselines, the hit rate can still be improved by nearly 100%.

Under the modes where the movement of the *data* is stochastic, the average hit rate of the RL-based method is higher than the IL-based method. The reason is that the IL-based method learns the behaviour of the player, but it cannot capture the randomness of the transformation relationship between the player and the *data*. However, the RL-based method models the player and the *data* together and catches the relationship between them to reduce the influence of the randomness.

D. Performance In the Real Mobile MMORPG

1) Experiment Settings: We take a specific quest as a test case to verify the performance of our proposed method with the game's built-in method which based on the rule and LRU in the real game environment. During the quest, the player will call for a lot of *data* and the cache capacity is set to 100. This specific quest will require players to complete a series of small quests.

Because this environment is a real game environment, the T_h^v is not explicitly accessible. Therefore, we use the RL-based method which takes the *data* in the cache, the requested *data*, the location of the player, the quest execution point as state, and the operations of the cache management as action. The reward function is defined as Equation 10, where α is 3000. And we use the IMPALA to train the model. Besides, we record the whole trajectory and compute the hit rate of Belady's Policy using this trajectory at the end of one episode. The results of Belady's Policy are also summarized in Table II for reference and it is worth noting that we cannot reach this upper limit in practice as it uses the future information in the whole trajectory at each decision-making step.

2) Results and Analysis: We conduct different methods to guide the cache management for this quest in the real game environment. And we test every method 200 times and calculate the average hit rate. The results are shown in Table II. It can be seen that the average hit rate of the RL-based method is 16% higher than the game's built-in method, which indicates that our cache management methods can adapt to the real game environments and outperform the game's built-in method.

V. RELATED WORK

Traditional cache management methods include LRU, FIFO and LFU, which are mainly focused on the cache replacement strategies. And some manually-engineered heuristics methods [14]–[17] are variations of the previous traditional methods. Some researches [18]–[22] take learning-based approaches to learn a cache management policy approximating the Belady's policy [4]. These methods perform well on some simple cache access patterns, but they perform poorly on complex cache access patterns because of the lack of knowledge from complex scenarios.

In recent years, some researches in the scenarios such as web caching and content caching consider the knowledge

of the scenario like content popularity and user preference. A large body of researches [23]–[27] focus on learning the content popularity distribution by rule-based mechanisms or machine learning to achieve near-optimal results. Some researches [28]–[30] seek the optimal caching policy by considering user preference together because the user preference will influence the requested data. In principle, the aforementioned approaches make decisions on cache based on information in application scenarios. And the mobile game as an application scenario closely related to players, taking the behavioural process of the players in games into consideration will help the cache management in selecting the right data to store in the cache.

VI. Conclusion

In this paper, we propose a future-oriented cache management (FOCM) to solve the cache management problem in mobile games. FOCM bridges the players' behaviours and the cache management's decisions with a dual Markov model, which consists of two Markov decision processes (MDPs). Based on this framework, we have put forward the imitation learning based (IL-based) method and reinforcement learning based (RL-based) method to infer the *data* that are most likely to be used in the future by learning the behaviour patterns of the players directly or indirectly. The experimental results have demonstrated that the proposed methods can achieve a higher hit rate in both the cache simulation environment and the real mobile MMORPG than other methods. To facilitate further research in this area, we release our simulated environment which includes Gym-like interfaces and can be easily called by various RL algorithms.

References

[1] P. Cohendet, D. Grandadam, C. Mehouachi, and L. Simon, "The local, the global and the industry common: the case of the video game industry," *Journal of Economic Geography*, vol. 18, no. 5, pp. 1045–1068, 2018.

[2] M. D. Griffiths and H. M. Pontes, "The future of gaming disorder research and player protection: what role should the video gaming industry and researchers play?" *International Journal of Mental Health and Addiction*, pp. 1–7, 2019.

[3] Newzoo, "2019 global games market report," 2019.

[4] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[6] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas, "Social influence as intrinsic motivation for multi-agent deep reinforcement learning," in *International Conference on Machine Learning*, 2019, pp. 3040–3049.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *arXiv preprint arXiv:1911.08265*, 2019.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[10] H. Daumé, J. Langford, and D. Marcu, "Search-based structured prediction," *Machine learning*, vol. 75, no. 3, pp. 297–325, 2009.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[14] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, "Improving cache management policies using dynamic reuse distances," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2012, pp. 389–400.

[15] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 180–193.

[16] H. Gao and C. Wilkerson, "A dueling segmented lru replacement algorithm with adaptive bypassing," 2010.

[17] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 60–71, 2010.

[18] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 175–186.

[19] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr, and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 430–441.

[20] A. Jain and C. Lin, "Back to the future: leveraging belady's algorithm for improved cache replacement," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 78–89.

[21] Z. Shi, X. Huang, A. Jain, and C. Lin, "Applying deep learning to the cache replacement problem," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 413–425.

[22] E. Z. Liu, M. Hashemi, K. Swersky, P. Ranganathan, and J. Ahn, "An imitation learning approach for cache replacement," *arXiv preprint arXiv:2006.16239*, 2020.

[23] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–6.

[24] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, 2018.

[25] J. Song, M. Sheng, T. Q. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Transactions on Communications*, vol. 65, no. 10, pp. 4309–4324, 2017.

[26] S. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching youtube content in a cellular network: Machine learning approach," *Ieee Access*, vol. 5, pp. 5870–5881, 2017.

[27] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[28] L. Lu, Y. Jiang, M. Bennis, Z. Ding, F.-C. Zheng, and X. You, "Distributed edge caching via reinforcement learning in fog radio access networks," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*. IEEE, 2019, pp. 1–6.

[29] Y. Jiang, M. Ma, M. Bennis, F.-C. Zheng, and X. You, "User preference learning-based edge caching for fog radio access network," *IEEE Transactions on Communications*, vol. 67, no. 2, pp. 1268–1283, 2018.

[30] Y. Jiang, M. Ma, M. Bennis, F. Zheng, and X. You, "A novel caching policy with content popularity prediction and user preference learning in fog-ran," in *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–6.