Received August 27, 2021, accepted October 4, 2021, date of publication October 15, 2021, date of current version October 25, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3120384

SpecMCTS: Accelerating Monte Carlo Tree Search Using Speculative Tree Traversal

JUHWAN KIM, BYEONGMIN KANG, AND HYUNGMIN CHO
Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Hyungmin Cho (hyungmin.cho@skku.edu)

This work was supported in part by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Korea Government [Ministry of Science, ICT (MSIT)] Research on CPU Vulnerability Detection and Validation under Grant 2019-0-00533, in part by the High-Potential Individuals Global Training Program under Grant 2020-0-01550, and in part by the Development of high Speed Encryption Data Processing Technology that Guarantees Privacy Based Hardware under Grant 2021-0-00779.

ABSTRACT Monte Carlo Tree Search (MCTS) algorithms show outstanding strengths in decision-making problems such as the game of Go. However, MCTS requires significant computing loads to evaluate many nodes in the decision tree to make a good decision. Parallelizing MCTS node evaluations is challenging because MCTS is a sequential process that each round of tree traversal depends on the previous node evaluations. In this work, we present *SpecMCTS*, a new approach for accelerating MCTS by speculatively traversing the search tree. Many MCTS applications, such as AlphaGo Zero, use a deep neural network (DNN) model to evaluate the tree nodes during the search. SpecMCTS uses a pair of DNN models, the *speculation model* and the *main model*. The faster (but less accurate) speculation model accelerates the sequential tree search while the more accurate main model improves the decision quality. SpecMCTS accelerates MCTS for the game of Go by up to 2.09× on the NVIDIA T4 GPU. This performance improvement can be translated into a better decision quality by performing a larger number of tree traversals within the time limit. For a fixed decision time, SpecMCTS shows stronger gameplay (higher win rate) than the original sequential MCTS and state-of-the-art MCTS parallelization approaches.

INDEX TERMS Monte Carlo Tree Search (MCTS), deep neural networks (DNNs), speculation, reinforcement learning.

I. INTRODUCTION

Monte Carlo Tree Search (MCTS) demonstrated its effectiveness in complex control domains that require future planning, such as video games [1] and the game of Go [2]–[4]. MCTS is typically applied to a Markov decision process setting where an agent (*e.g.*, an AI game player or an autonomous driving program) makes decisions in the target environment based on the current state. In MCTS, an agent "*looks ahead*" the future scenarios by traversing the search tree. In the search tree, each node corresponds to a state and edges represent actions performed by the agent from the state. After performing the look-ahead tree traversal many times, the agent selects the best action predicted by the tree traversal.

For a complex decision-making process, exploring the entire tree is infeasible. For example, in Go, the search tree represents each move of the black or the white stone on the Go board that has 19 × 19 possible move positions and a game episode typically consists of more than 200 moves. Therefore, the key objective of an MCTS algorithm is guiding the tree traversal (*i.e.*, narrowing the search tree) so that the agent can selectively visit more important nodes to make a good decision.

To select the next child to visit in the search tree, MCTS algorithms, such as the upper confidence bound for trees (UCT) [5], evaluate each node (details in Sec. II). Evaluating a node state can be done by Monte-Carlo rollouts, which sample a possible outcome from the state by "playing" or "simulating" the state using the environment simulator for a certain period (or until the end of an episode). Each step of actions performed by the agent during the rollout simulations is usually determined using simplified computations than the MCTS itself, such as random actions or a small neural network (*e.g.*, the original AlphaGo [2]). Although the rollout simulations require considerable computing loads, a node evaluation obtained through the rollout is just a sample of

The associate editor coordinating the review of this manuscript and approving it for publication was Valentina E. Balas.

the estimated outcome. AlphaGo Zero improves upon the original AlphaGo by eliminating such Monte-Carlo rollouts. Instead, in AlphaGo Zero, a node is evaluated using a deep neural network (DNN). The DNN model for the tree node evaluation is trained using human-generated datasets from professional players or datasets generated through self-play between the reinforcement learning agents without human knowledge.

Regardless of the node evaluation method (whether it is based on Monte-Carlo rollout simulations or calculated using DNNs) the dominant computing loads of MCTS come from those node evaluations, rather than the tree traversing. Therefore, to accelerate MCTS, either the time for a node evaluation has to be shortened or multiple nodes have to be evaluated in parallel. Reducing the node evaluation time through approximation may result in poor decision results. Parallelizing the node evaluations is not a trivial problem because MCTS is inherently sequential. During the search process using MCTS, each tree traversal depends on previous node evaluations.

There were many attempts to parallelize the MCTS process [6]–[8], but most of the parallelization approaches result in decision quality degradation. To evaluate multiple nodes in parallel, the MCTS algorithm may end up including less important nodes in the evaluation process as well.

In this work, we present a new MCTS acceleration approach, called *SpecMCTS*, that balances the performance and decision quality. SpecMCTS targets MCTS applications that use a DNN model to evaluate a state, such as AlphaGo Zero. SpecMCTS accelerates the search process by using a pair of DNN models: the *speculation model* and the *main model*. These models are trained for the same objective functions, but they use different DNN configurations to be used as different roles during the tree traversal.

The speculation model is a smaller, but a faster model that approximates the node evaluation results to quickly guide the next tree traversal. The main model is a full-size DNN model that aims for higher accuracy. When evaluating a node, SpecMCTS schedules the inference computations for both DNN models at the same time. As soon as the speculation model completes, the agent starts the next tree traversal based on the approximated (*i.e.*, *speculated*) node evaluation. Although the speculation model may result in less accurate node evaluations, the resulting decision quality is better than the previous state-of-the-art for MCTS acceleration. The main model is processed in a separate thread while the MCTS agent is advancing the search process ahead using the speculation model. Once the main model completes, the updated node evaluations are reflected in the search tree. After the update, subsequent tree traversals are guided by the more accurate node evaluations.

Many modern GPUs or neural processing units (NPUs) achieve higher throughput with larger computation batches or parallel execution of multiple threads [9], [10]. The original MCTS that performs one node evaluation at a time (*i.e.*, *sequential MCTS*) under-utilizes such GPU or NPU

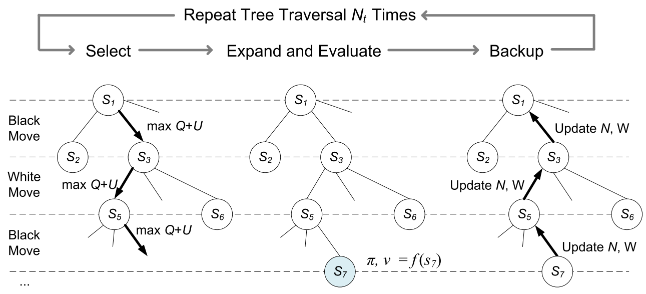


FIGURE 1. MCTS tree traversal phases.

platforms. SpecMCTS exploits the computational capacity of GPUs by executing multiple inference tasks concurrently.

In this paper, we evaluate the performance and the decision quality of SpecMCTS for the game of Go. Compared to the sequential MCTS, SpecMCTS accelerates the tree traversal process by up to 2.07× on the NVIDIA Tesla T4 GPU. When the MCTS players are limited to a fixed decision time, SpecMCTS can result in a higher win rate compared to the sequential MCTS and the previous state-of-the-arts for MCTS acceleration.

II. BACKGROUND: MCTS IN AlphaGo ZERO

To determine the next action, the MCTS search executes multiple rounds of tree traversals (*i.e.*, looking ahead the future action sequences). Each node s in the search tree has outgoing edges (s,a) for each of the possible actions. In a board game like Go where two players alternately perform actions, each level in the search tree represents the alternating player's actions. During MCTS, the agent also predicts the opponent's moves (actions) assuming both players are trying to maximize the possibility to win the game.

In AlphaGo Zero, a node evaluation calculates two statistics: the value v and the policy π. The value is a scalar value $v \in \{-1, +1\}$ that represents the expected outcome of the game, where $+1$ is a win by the black stone and -1 is a win by the white stone. The policy is a probability distribution that suggests an action out of the possible moves (19×19 positions of the Go board plus an extra *pass* action). An agent may directly use this raw policy output π to determine the next action without MCTS, but it results in inferior decision quality than MCTS [3]. The MCTS process enhances decision quality over the raw policy. Through the tree traversals, the agent builds more concrete predictions on future outcomes and strengthens its action planning.

AlphaGo Zero uses a variant of the PUCT algorithm [11] to guide the search traversal. Each edge in the tree maintains the following set of statistics: $N(s, a)$, $W(s, a)$, and $P(s, a)$, where $N(s, a)$ is the visit counts, $W(s, a)$ is the accumulated state values of all successor nodes that have been explored through the current node, and $P(s, a)$ is the raw policy $\pi(a)$.

An MCTS agent repeatedly performs tree traversals, and each tree traversal consists of the following three phases (Fig. 1):

1) **Select:** The search starts from the current root of the tree and traverses down the tree. At a node s_t, the search process selects the next edge a_t according to the statistics stored in the search tree as follows:

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + U(s_t, a))$$
$$Q(s, a) = \frac{W(s, a)}{N(s, a)}$$
$$U(s, a) = C_{puct} P(s, a) \frac{\sqrt{\Sigma_b N(s, b)}}{1 + N(s, a)}$$

   Q guides the agent to edges with a high average state value (*exploitation*) and U guides the agent to edges with a high policy probability and a low visit count (*exploration*). C_{puct} is a constant that determines the level of exploration.
2) **Expand and Evaluate:** When the tree traversal encounters an edge that never visited before, the search process adds a new successor node (node s_7 in Fig. 1) and evaluates the node. AlphaGo Zero evaluates the new state through a DNN inference, using the history of the Go board status as the input to the DNN model.
3) **Backup:** To incorporate the new node evaluation in the search tree, the visit counts (N) and the state value accumulations (W) are updated along the reverse path from the new leaf node to the root.

The tree traversal continues until the agent completes the requested number of tree traversals or until the expiration of the decision time limit. After the MCTS finishes, the agent determines the next action to play. In the evaluation mode (*i.e.*, when not used for training), AlphaGo Zero selects the action with the largest visit counts.

III. RELATED WORK

A. PRIOR WORK ON PARALLELIZING MCTS

Evaluating multiple tree nodes in parallel can accelerate the MCTS process. To select multiple nodes, the basic sequential MCTS algorithm explained in the previous section has to be modified. Otherwise, an MCTS agent always traverses the search tree through the same path and selects the same node unless a new node evaluation result is reflected in the tree.

Earlier approaches for MCTS parallelization select multiple nodes at the root of the tree search (*root parallelization*) or multiple nodes at the leaf level (*leaf parallelization*) [6], [12]. However, these simple approaches do not perform well on a large search space, such as the game of Go.

AlphaGo Zero uses a technique called the *virtual loss* to divert the agent to a different path [3]. The virtual loss temporarily evaluates the selected node as having a negative value that corresponds to a loss of the game. This temporarily added virtual loss value is reverted after the actual node evaluation is calculated. Since the virtual loss is applied all edges along the path to the root, the next tree traversal would be less inclined to take the same path and more likely to choose a different node. The virtual loss technique shows better results than the root parallelization or the leaf parallelization. However, the decision quality is usually worse than the sequential MCTS when the same number of tree traversals are executed.

WU-UCT is recently proposed as another variation of the UCT algorithm for parallel MCTS [8]. WU-UCT uses a different formula for selecting an edge when traversing the tree. WU-UCT algorithm considers an extra node statistics $O(s, a)$ that reflects the number of evaluation threads that are working on the edge.

B. MCTS ACCELERATION WITHOUT PARALLELIZATION

MCTS acceleration mechanisms that do not use parallelization have been studied as well.

MPV-MCTS combines two neural networks of different sizes [13]. Although the use of two neural networks is similar to our approach, MPV-MCTS does not parallelize the node evaluations. MPV-MCTS sequentially evaluates the nodes by alternatively using the larger and the smaller DNN models within the time budget.

DS-MCTS reduces the search time by terminating the search process prematurely based on the uncertainty of the current state [14]. Such early termination can accelerate the self-play training process by reducing the average search time. However, in many decision-making situations, the time for making a decision is often fixed, and obtaining a higher-quality decision within the time limit is more important than completing a single search fast.

IV. LIMITATIONS OF PARALLELIZED MCTS

To provide an intuition why a parallelized MCTS may result in lower decision quality, we compare the timeline of the sequential MCTS and the parallel MCTS in Fig. 2 (a) and (b). In the figure, the node (state) indices are numbered based on the order the nodes are selected for the evaluation. For example, s_1 is the first node selected for evaluation and s_2 is the second node selected for the evaluation. The evaluation order is determined by the MCTS tree traversal, and the order is not the absolute ID of the node. For each MCTS search, s_n may represent different nodes in the search tree.

The white rounded boxes in Fig. 2 represent the MCTS select phases, and the numbers in the angle brackets represent the range of the evaluated nodes that were reflected in the tree statistics at the time of the next node selection. For example, the third white box of the sequential MCTS (Fig. 2 ❶) depicts the select phase for choosing the third node (s_3) to evaluate. At this moment, the MCTS tree *knows* the evaluation results of the previously selected nodes s_1 and s_2. Therefore, the node selection for s_3 can be made upon the knowledge accumulated up to s_2. Fig. 2 ❶ denotes this select phase as "Select ⟨2⟩".

When parallelizing MCTS, however, each node selection might not be based on the complete knowledge that covers all previously selected nodes. In Fig. 2 (b), the select phases are repeated before the node evaluations to find multiple nodes for an inference batch. The figure shows an example where the batch size is three. In this case, a select phase is forced to make a node choice without knowing the evaluation results of the other nodes in the same inference batch.

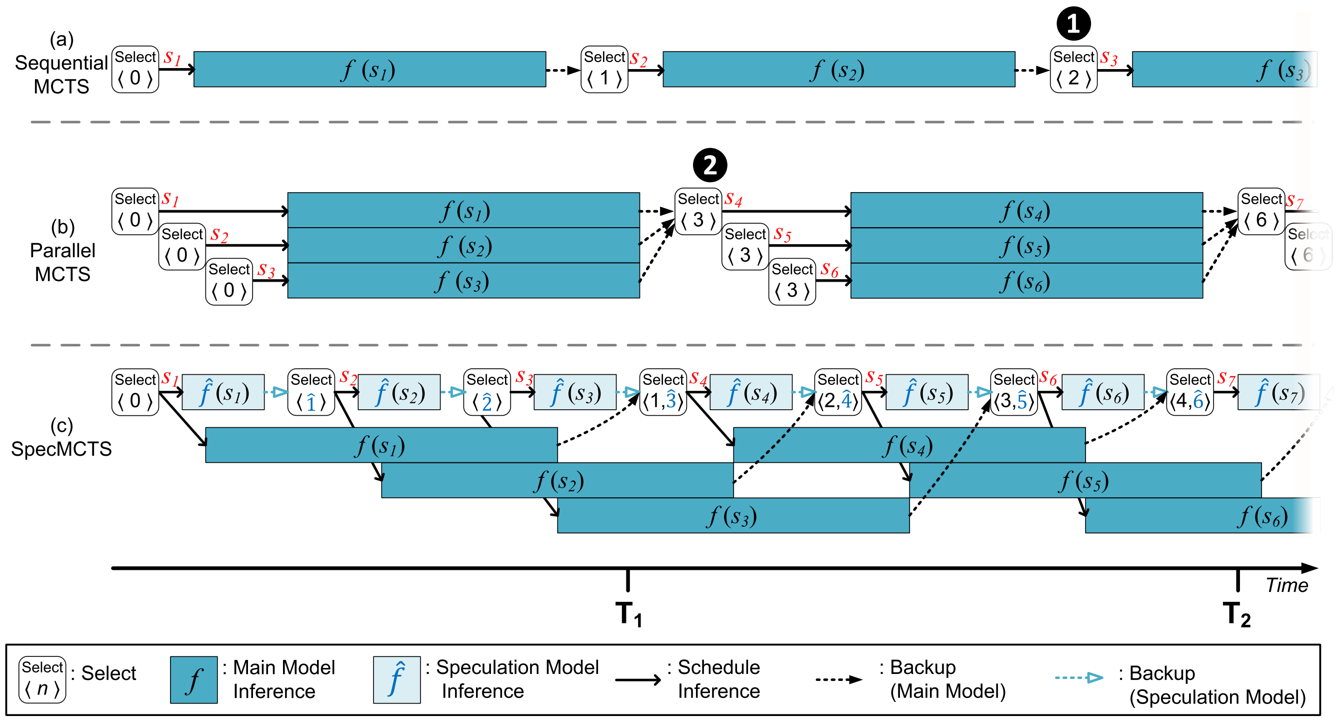


FIGURE 2. **Comparison of MCTS tree traversal timelines.**

When selecting the second batch for inference (Fig. 2 ❷), s_4, s_5, and s_6 are chosen from a tree where the node evaluations only up to s_3 are reflected (*i.e.*, "Select ⟨3⟩"). In the sequential MCTS, s_6 would have been selected on a tree where s_4 and s_5 evaluations are reflected (*i.e.*, "Select ⟨5⟩"). Therefore, the quality of the node selections by the parallel MCTS could be inferior to that of the sequential MCTS.

More seriously, this sub-optimality accumulates as the tree traversal continues. If the previous inference batch contained less important nodes, the subsequent node selections are made upon the less valuable information. This chain of sub-optimal selections continues until the end of the game episode since the MCTS agent does not discard the search tree after deciding an action. The search tree with accumulated node evaluation statistics is reused for selecting the next move.

V. SPECULATIVE TREE TRAVERSAL BY SpecMCTS

A. SpecMCTS OVERVIEW

SpecMCTS accelerates MCTS while complying with the sequential nature of the tree traversal. Using two separate DNN models, SpecMCTS decouples the sequential tree traversal from the accurate node evaluations that require longer computation time. The agent traverses the tree similar to the sequential MCTS, but based on the approximated node evaluations ($\hat{\pi}$ and $\hat{v}$) obtained using the speculation model ($\hat{f}$).

Since $\hat{\pi}$ and $\hat{v}$ are less accurate than the outputs from the main model (f), the selected nodes are concurrently evaluated using the main model as well. When the slower main model finishes the more accurate node evaluations (*i.e.*, π and v), the differences are corrected in the tree statistics. Similar to the backup phase in the sequential MCTS, the updates are also applied to the nodes along the reverse path to the root node.

The main model inferences happen in different threads using separate GPU streams. Therefore, the long inference time of the main model does not become a direct bottleneck for the tree traversal as long as the computation platform has enough capacity to perform multiple inferences simultaneously. On a platform with a limited computing capacity, we may throttle the speculation model execution by checking the main model inference waiting queue in order to prevent the main model evaluation results from being updated too late.

Figure 2 (c) shows the timeline of SpecMCTS. The select phases of SpecMCTS also indicate the range of speculated node evaluations and main node evaluations reflected in the search tree. For SpecMCTS, there are two numbers in the bracket to distinguish the speculated node evaluations from the node evaluations completed by the main model. "Select $\langle a, \hat{b} \rangle$" means that the node evaluations are completed up to the b-th node using the speculation model. Only the evaluations up to the a-th node have been corrected with the main model results. Usually, $a < b$ unless the GPU switches the execution order. We set a higher priority to the GPU stream for the speculation model.

Figure 3 exemplifies the difference between the parallel MCTS and SpecMCTS by depicting example search tree instances. Figure 3 corresponds to the search phase in

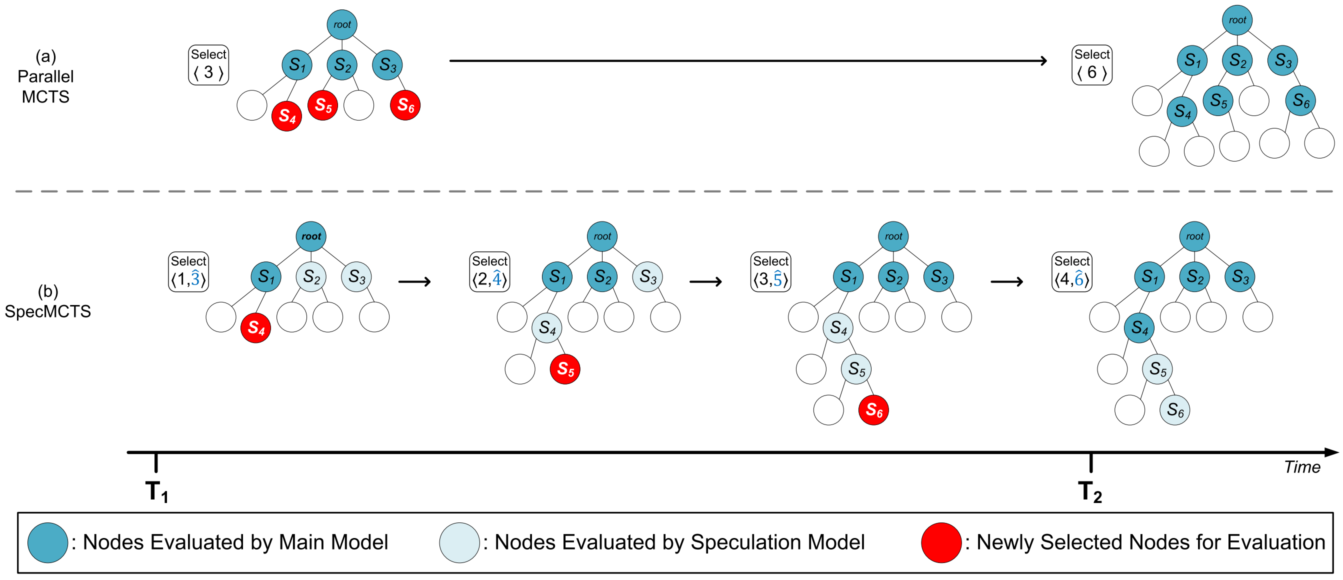


FIGURE 3. An example search tree state comparison during MCTS.

between T_1 and T_2 of Fig. 2. SpecMCTS selects s_4 from a tree where the node evaluations up to s_3 are reflected. Although only node s_1 has been corrected with the main model results, the search process can follow the similar tree traversal as the sequential MCTS if the speculation model results are close enough to the main model results. The next node selection, s_5, is made on a tree *after* the node s_4 is evaluated. Also, by this time, node s_2 is also corrected with the main model results. Similarly, s_6 is selected upon a search tree with more node evaluation results. In contrast, the parallel MCTS selects all of s_4, s_5, and s_6 from a search tree with node evaluations only up to s_3. Compared to the parallel MCTS, each node selection by SpecMCTS can be made out of a richer knowledge on the search tree.

Algorithm 1 Sequential MCTS

```
Input: root node $s_r$, main model $f$, number of traversals $N_t$
1  $s_t \leftarrow s_r$
2  for {1..$N_t$}
3      while $s_t$ is an expanded node                          ▷ Select
4          $a_t = \operatorname{argmax}_a(Q(s_t, a) + U(s_t, a))$
5          $s_t \leftarrow$ child of $s_t$ on edge $a_t$
6      $\pi, v \leftarrow f(s_t)$                                 ▷ Evaluate
7      $P(s_t) \leftarrow \pi$, and mark $s_t$ as an expanded node.  ▷ Expand
8      while $s_t$ is not $s_r$                                  ▷ Backup
9          $a_t \leftarrow$ edge from the parent of $s_t$, and $s_t \leftarrow$ parent of ($s_t$)
10         $W(s_t, a_t) \leftarrow W(s_t, a_t) + v$, and
           $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
11 return $\operatorname{argmax}_a(N(s_r, a))$
```

B. SpecMCTS ALGORITHM

Algorithm 1 describes the original MCTS algorithm. The algorithm traverses the search tree N_t times (line 2). Within each iteration, the agent performs the Select, Evaluate, and the Backup steps as depicted in Fig. 1. The Select process traverses the tree downwards from the root node (s_r) until it finds a node that has not been expanded (lines 3-5). While doing so, the agent selects the node with the highest $Q + U$ value at each level (line 4). The selected node is evaluated using a DNN inference (line 6) and the node is indicated as an expanded node (line 7). The obtained value from the node evaluation is reflected in the tree by traversing upwards to the root node (lines 8-10). After N_t traversals are finished, the agent selects the edge (*i.e.*, action) with the most visit count (line 11).

The modified parts for SpecMCTS are highlighted with an underline in Algorithm 2. The agent schedules the main model inference to be computed in the background (line 6) and then waits for the results from the faster speculation model (line 7). Lines 8 and 12 show that the speculated node evaluations are reflected in the search tree ahead of the main evaluation results. SpecMCTS has an additional node statistics $\hat{V}$ that saves the value calculated by the speculation model (line 9). $\hat{V}$ will be used by Algorithm 3 to calculate the difference with the value from the main model (Δv). When the evaluation using the main model completes, Algorithm 3 corrects the differences in the search tree.

SpecMCTS resembles the speculative execution in pipelined microprocessors. While the real outcome of a branch instruction is being resolved, the processor speculatively executes the next instructions based on the branch predictor's quick decision. Unlike branch prediction in microprocessors, SpecMCTS does not discard the evaluation

Algorithm 2 SpecMCTS

```
Input: root node $s_r$, speculation model $\hat{f}$, main model $f$,
       number of traversals $N_t$
1  $s_t \leftarrow s_r$
2  for {1..$N_t$}
3      while $s_t$ is an expanded node                        ▷ Select
4          $a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + U(s_t, a))$
5          $s_t \leftarrow$ child of $s_t$ on edge $a_t$
6      Schedule $f(s_t)$                     ▷ Asynchronous main evaluation
7      $\hat{\pi}, \hat{v} \leftarrow \hat{f}(s_t)$                   ▷ Speculative evaluation
8      $P(s_t) \leftarrow \hat{\pi}$, and mark $s_t$ as an expanded node.  ▷ Expand
9      $\hat{V}(s_t) \leftarrow \hat{v}$                            ▷ Save $\hat{v}$
10     while $s_t$ is not $s_r$                                ▷ Backup
11         $a_t \leftarrow$ edge from the parent of $s_t$, and $s_t \leftarrow$ parent of ($s_t$)
12         $W(s_t, a_t) \leftarrow W(s_t, a_t) + \hat{v}$, and
           $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
13 return $\underset{a}{\operatorname{argmax}}(N(s_r, a))$
```

Algorithm 3 Updating Main Model Results

```
Input: $\pi$ and $v$ from $f(s_t)$, evaluated node $s_t$, root node $s_r$
1  $P(s_t) \leftarrow \pi$                            ▷ Replace $\hat{\pi}$ with $\pi$
2  $\Delta v \leftarrow v - \hat{V}(s_t)$          ▷ Difference between $\hat{v}$ and $v$
3  while $s_t$ is not $s_r$                         ▷ Backup delta
4      $a_t \leftarrow$ edge from the parent of $s_t$, and $s_t \leftarrow$ parent of ($s_t$)
5      $W(s_t, a_t) \leftarrow W(s_t, a_t) + \Delta v$
```

results of wrongfully selected nodes because those node evaluations also help to build the knowledge of the current state.

VI. CONSTRUCTING THE SPECULATION MODELS

The speculation model should be a lightweight model that closely matches the output of the main model. A speculation model can be created in many different ways, such as DNN pruning, quantization, or matrix factorization [15]–[17]. Although SpecMCTS is compatible with any DNN model compression technique, the compression for the speculation model should be aggressively geared towards having a faster inference time rather than preserving the inference accuracy.

In this paper, we construct the speculation models using two different approaches. One of the approaches is training a DNN model that has a smaller DNN structure, and the other is punning a trained network to reduce the inference time.

A. TRAINING A SMALLER DNN MODEL

Figure 4 depicts AlphaGo Zero's DNN model. The model starts with a convolution layer, followed by repeated residual blocks. Each residual block has two convolution layers plus a residual addition layer. After repeating the residual block 19 times, the network diverges into two different branches for the value and policy outputs. Each branch has another convolution layer, followed by fully-connected layers to produce the outputs.

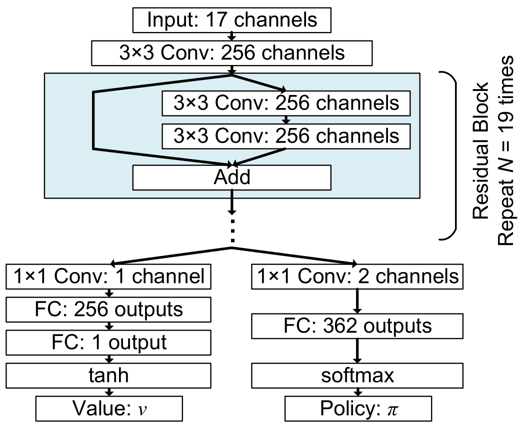


FIGURE 4. The DNN model structure ofAlphaGo Zero.

In this smaller model approach, we change the number of residual blocks to construct lightweight speculation models. Instead of the 19 residual blocks in the main model, the speculation models use 4, 2, or 1 residual blocks. We train the speculation models using the Knowledge Distillation technique [18], where the main model acts as the teacher model.

AlphaGo Zero is originally designed for reinforcement learning that generates the training data through self-play. In this work, we did not generate our own dataset to focus on the performance improvements and the decision quality differences when SpecMCTS is applied to actual gameplay situations rather than the training phase. Instead, to train the speculation model, we used a publicly available training dataset shared by MiniGo.[1] The MiniGo dataset contains more than a billion training samples obtained through the self-play procedure.

We trained the speculation models using the same hyper-parameters used for AlphaGo Zero as follows:

- Optimizer: Stochastic gradient descent (momentum set to 0.9)
- Learning rate: Start from 10^{-2}, $\times$ 1/10 in every 200K steps.
- Mini-batch size: 2,048
- L2 regularization parameter: $c = 10^{-4}$

B. PRUNING A TRAINED DNN MODEL

DNN pruning is one of the popular ways to construct a lightweight model. The pruning approaches can be classified into filter pruning [19], [20] and weight pruning [15], [21], [22]. Filter pruning finds less important filters (*i.e.*, channels) and eliminates the whole filter. In contrast, the granularity of weight pruning is individual weight values in a filter. Although weight pruning generally provides better accuracy

[1] `gs://minigo-pub/v13-19 × 19/data/`

TABLE 1. Comparison of the main and the speculation models.

Model	Configuration	FLOPs	Inference time (ms)
Main model	Blocks: 19, Channels: 256	16.31G	1.25
Speculation model (fewer blocks)	Blocks: **4**, Channels: 256	3.53G	0.31
	Blocks: **2**, Channels: 256	1.83G	0.18
	Blocks: **1**, Channels: 256	0.97G	0.12
Speculation model (channel pruning)	Blocks: 19, Channels: **90**	2.02G	0.32
	Blocks: 19, Channels: **32**	0.26G	0.19
	Blocks: 19, Channels: **16**	0.07G	0.17

with higher compression rates, the obtained performance improvement may not be as high as the improvement from filter pruning. To skip individual weight elements, the computing hardware must support such sparse computations. On GPUs, such computation with sparse filters has limited support. For example, the tensor cores in the NVIDIA Ampere architecture support sparse computations, but the maximum throughput gain is only up to 2×, and the weight values have to be pruned in a structured way. In this work, we tested a filter pruning approach called HRank [23] for an aggressive execution time reduction. HRank selects feature maps based on their rank to find filters to prune.

C. COMPARISON OF SPECULATION MODEL INFERENCE TIME

Table 1 compares the characteristics of the main model and the speculation models. The FLOPs (floating point operations) count is based on the conventional 2D convolution algorithm. The inference latency is measured on the NVIDIA Tesla T4 GPU using the TensorRT inference engine [24].

In the smaller model approach, the ratio of layer reduction is almost analogous to the execution time reduction. When we reduce the number of residual blocks from 19 to 4, the inference time is reduced by 4×.

To obtain a similar execution time improvement, the pruning approach has to use a higher compression rate. This is because we cannot change the number of layers in filter pruning, and the layers have to be executed sequentially. For instance, at a similar inference time of around 0.3 ms, the speculation model with 4 residual blocks and 256 channels performs 3.53 GFLOPs. On the contrary, the pruned model with 90 channels can conduct 2.02 GFLOPs of computations only. The difference becomes larger on smaller models.

Due to this limitation, if both models have a similar inference latency, the models obtained through the weight pruning approach may have lower decision quality than the smaller model trained from scratch. In Section VII-B, we evaluate one of the speculation models obtained using the pruning approach and confirmed that its decision quality is lower than that of the speculation model obtained using the smaller model approach.

This comparison does not mean that pruning approaches are worse than the smaller model approach. This result implies that SpecMCTS is sensitive to the quality and performance trade-offs of the speculation model and a wider variety of approaches to construct the speculation model has to be evaluated as future work.

VII. EVALUATION

We implemented SpecMCTS on an open-source AlphaGo Zero replica [25]. During the search process, DNN inference tasks are performed using NVIDIA TensorRT (version 7.2.2.3), which is a GPU runtime optimized for DNN inference [24]. For a better inference performance, all models are optimized to FP16 computations to utilize the Tensor Cores in the GPU.

We tested the quality of the MCTS algorithms by performing many Go game matches between the compared MCTS algorithms and calculated the win rate. To collect a large number of game samples (more than 1,000 matches per each win rate calculations), we utilized Google Cloud Platform's Tesla T4 GPU instances.

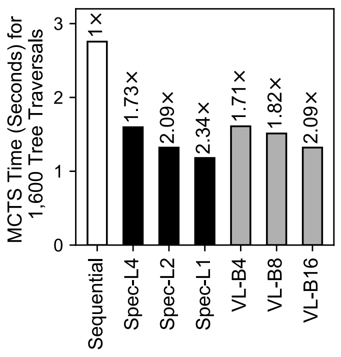
In this evaluation, we compare the following MCTS configurations:

1) **Sequential**: The original sequential MCTS.
2) **Spec-L***n*: SpecMCTS configuration that uses a speculation model with *n* residual blocks.
3) **Spec-Pr**: SpecMCTS configuration that uses a speculation model obtained using the filter pruning approach. We test only a single configuration (channels: 90) for Spec-Pr.
4) **VL-B***n*: Parallel MCTS configuration with the virtual loss technique where *n* node evaluations form an inference batch to be computed together.
5) **WU-B***n*: Parallel MCTS with the batch size of *n*. Unlike VL-B*n*, the tree traversal is guided by the WU-UCT algorithm instead of AlphaGo Zero's PUCT algorithm.

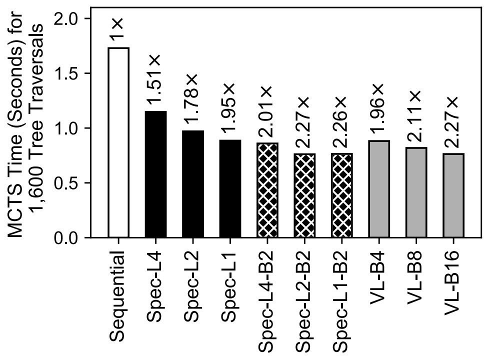
A. MCTS TIME

Figure 5 compares the execution time of different MCTS configurations for selecting an action by performing 1,600 tree traversals. On the T4 GPU (Fig. 5a), the sequential MCTS takes more than 2.7 seconds to perform 1,600 tree traversals. The GPU computing resources are under-utilized with the sequential MCTS.

On the T4 GPU, Spec-L4 reduces the execution time by 1.73× compared to the sequential MCTS. The SpecMCTS



(a) NVIDIA Tesla T4



(b) NVIDIA A100

FIGURE 5. MCTS search time comparison for different MCTS configurations. Performance gains are indicated in the bar labels.

search time improvement is lower than the execution time improvement of the speculation model alone. This is because the speculation model and the main model are executed on the same GPU in this evaluation. The performance of SpecMCTS can be improved by using a faster speculation model. By using smaller speculation models, Spec-L2 and Spec-L1 increase the performance gain to 2.09× and 2.34×, respectively.

The performance improvements of VL-B*n* configurations are affected by the inference batch size. On the T4 GPU, when the batch size is increased from one (Sequential) to four (VL-B4), the search process is accelerated by 1.71×. However, the gain diminishes after the batch size is increased beyond 8. WU-B*n* models are not shown in Fig. 5 because their execution time is almost identical to VL-B*n* models with the same batch size.

We also tested the scalability on a larger GPU, NVIDIA A100 (Fig. 5b). On the A100 GPU, the MCTS time of all configurations becomes shorter than their MCTS time on the T4 GPU. However, the relative performance gains of Spec-L*n* models over the sequential MCTS do not get better on the A100 GPU compared to the gains on the T4 GPU. The speed-up of SpecMCTS is tied to the inference time difference of the main model and the speculation model. On the contrary, VL-B*n* models show a slightly better performance on a larger GPU. For instance, VL-B4's performance gain is increased from 1.71× on Tesla T4 to 1.93× on A100.

We can use a hybrid approach of SpecMCTS and the virtual loss technique to make the SpecMCTS models have comparable performance to VL-B*n* models on a larger GPU. **Spec-L*n*-B*k*** models use a small degree (k) of parallelism when the speculation model is evaluating the search tree nodes. For example, in the Spec-L4-B2 configuration, the speculation model evaluates two nodes in parallel to schedule two main model inferences at the same time. This enables the SpecMCTS models to utilize the higher computing capacity of a bigger GPU platform. Spec-L*n*-B2 models achieve simliar performance gain as VL-B*n* models on the A100 GPU.

B. WIN RATE COMPARISON FOR THE SAME NUMBER OF TREE TRAVERSALS

MCTS acceleration approaches deviate from the sequential MCTS and result in decision quality degradation for the same number of tree traversals. Figure 6 compares the win rate of the MCTS configurations when they play matches against the sequential MCTS.

For the Spec-L*n* configurations, the win rate becomes lower as the speculation model gets smaller. The win rate of Spec-L4 agsinst the sequential MCTS is very close to 50.0%, which means the decision quality is almost not degraded from the sequential MCTS. The win rate with smaller models are lower than 50%. For example, when the number of tree traversal per decision (N_t) is 100, the win rate of Spec-L1 is degraded to 32.7%. However, the win rate of SpecMCTS is higher than the win rate of the VL-B*n* configurations with a similar inference time. VL-B16, which has a similar MCTS search time as Spec-L1, results in a win rate of 25.6% when N_t 100. Similarly, across all N_t, Spec-L4's win rate is higher than VL-B4's win rate, Spec-L2's win rate is higher than VL-B8's, and Spec-L1's win rate is higher than VL-B16's.

The win rate gap between the Spec-L*n* models and the VL-B*n* models diminishes as N_t increases. When N_t is 100, Spec-L4 has a 6% higher win rate than VL-B4, but the win rate of both models are almost similar when N_t is 1,600. The win rates of both models are close to 50%, meaning that they have similar decision quality as the sequential MCTS when N_t is large. A large number of tree traversals can alleviate the degradation of the node selection quality. An agent needs to carefully select important nodes (*i.e.*, more probable future scenarios) for better decision quality. If the decision time is short, each node selection highly matters. On the contrary, if the decision time is long, the decision quality is less sensitive to each node selection since the agent can visit a sufficient number of important nodes to evaluate. Therefore, if the decision time gets long enough, the differences between the MCTS algorithms diminish.

As aforementioned, the speculation model obtained using filter pruning has lower decision quality than Spec-L*n* configurations that have a similar execution time. Spec-Pr, which has a similar performance as Spec-L4, shows lower win rates than Spec-L4's win rates.

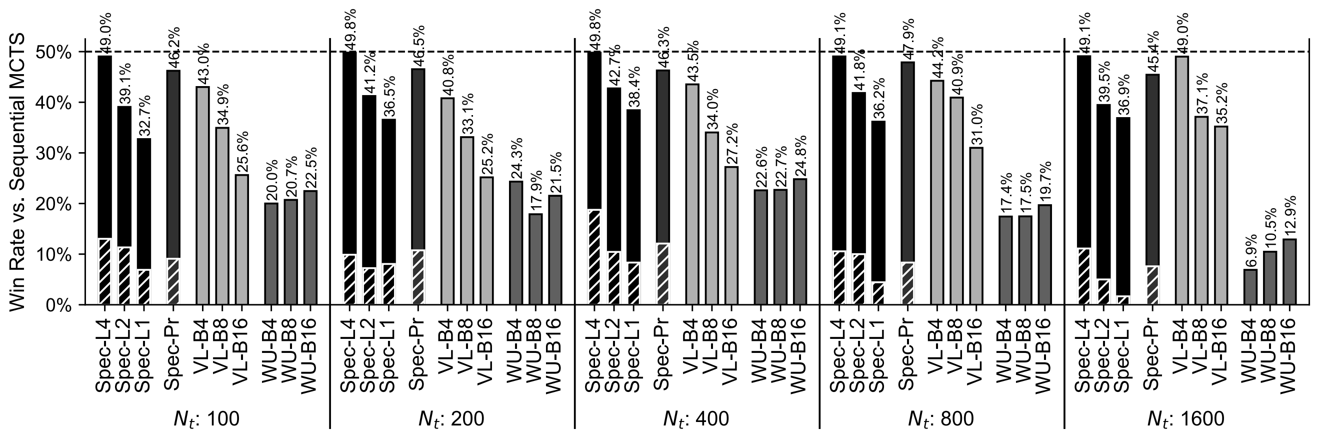


FIGURE 6. Win rate comparison for a fixed number of tree traversals (N_t).

TABLE 2. Number of tree traversals executed within the fixed decision time limit of 1 second.

NVIDIA Tesla T4	Sequential	Spec-L4	Spec-L2	Spec-L1	VL-B4	VL-B8	VL-B16
	580	1,000	1,208	1,352	992	1,056	1,200
NVIDIA A100	Sequential	Spec-L4-B2	Spec-L2-B2	Spec-L1-B2	VL-B4	VL-B8	VL-B16
	924	1,860	2,052	2,092	1,812	1,952	2,096

In SpecMCTS, the inaccurate node evaluations from the speculation model are corrected later by the main model. Without such correction by the main model, the speculation models alone cannot achieve a good decision quality. The hatched bars in the lower part of Spec-Ln graphs in Fig. 6 indicate the win rate when the agent only uses the speculation model without the main model. For instance, when N_t is 1,600, if the speculation model of Spec-L4 is executed alone, the obtained win rate is only 11.1%.

The WU-Bn models generally show lower win rates. Please note that in [8], the WU-UCT algorithm is designed for a different setting where Monte-Carlo rollout simulations are used instead of using a DNN inference for node evaluations.

C. WIN RATE COMPARISON FOR THE CONSTANT AMOUNT OF DECISION TIME

A more realistic scenario of MCTS application is each player is given a constant amount of time for the decision rather than fixing the number of tree traversals. In such a case, the higher performance of MCTS algorithms can be turned into a better decision quality by performing a larger number of tree traversals within the time limit. Table 2 lists the number of tree traversals within the fixed time limit of 1 second. On Tesla T4, the sequential MCTS can execute only 580 tree traversals for 1 second of decision time. Within the same time limit, Spec-Ln and VL-Bn configurations can execute a much larger number of tree traversals. The number of tree traversals is decreased proportionally for a shorter decision time.

In Figs. 7 and 8, we evaluate the relative win rate between the MCTS configurations when the decision time is fixed. We compare seven MCTS configurations: Sequential, three Spec-Ln (Spec-Ln-B2 for A100), and three VL-Bn. Unlike the comparison in Fig. 6, where all MCTS configurations play matches against the sequential MCTS only, all possible match configurations (*i.e.*, $\binom{7}{2}$ combinations) are tested, and the overall win rate is calculated. Therefore, the win rates in Figs. 7 and 8 represent the relative strengths of the configurations.

When the decision time is fixed, the sequential MCTS performs poorly because it can only execute a much smaller number of tree traversals. For example, in Fig. 7, when the decision time is 1 second on the T4 GPU, the win rate of the sequential MCTS is only 37.4%. The win rate of the sequential MCTS is especially poor on longer decision time because the sequential MCTS can explore only a small number of nodes while other configurations can explore enough number of tree nodes.

On the T4 GPU, the SpecMCTS configurations constantly show higher win rates than other configurations. Also, in all time limit cases, the best performing configuration is one of the SpecMCTS configurations. As discussed in the previous subsection, SpecMCTS is more effective when a quick decision is required. On the T4 GPU, when the decision time is limited to 1/8 of a second, the average win rate of Spec-Ln is 57.1% while the average win rate of VL-Bn is 43.1%.

On the A100 GPU, the SpecMCTS models still exhibit strong decision quality up to the decision time of 1/2 of a second. This means that the SpecMCTS approach is scalable to a platform with a large computing capacity as well if we employ the hybrid approach (Spec-Ln-Bk). However, when a decision time of 1 second is allowed, the differences between

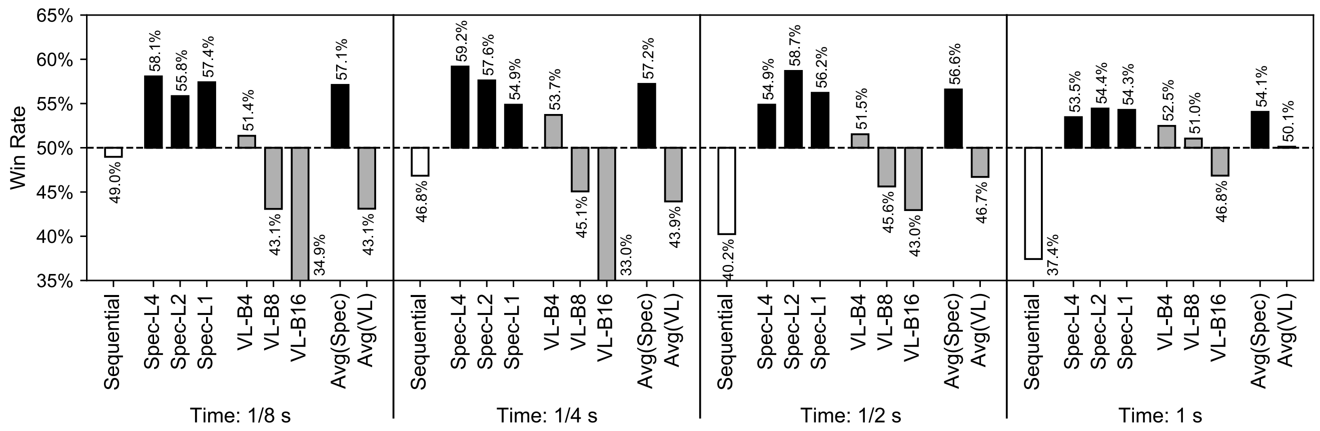


FIGURE 7. Win rate comparison for a fixed decision time limit on the NVIDIA Tesla T4 GPU.

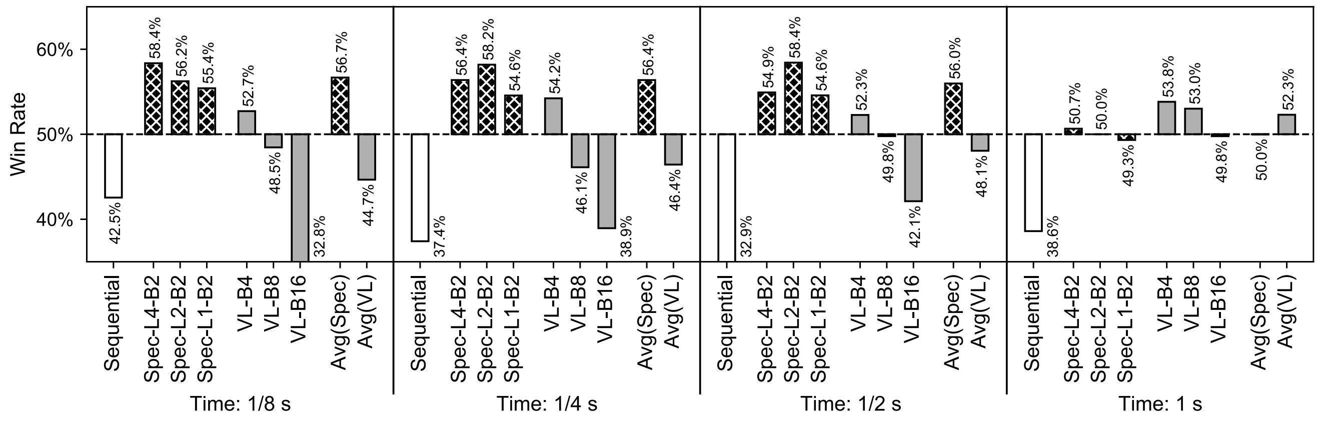


FIGURE 8. Win rate comparison for a fixed decision time limit on the NVIDIA A100 GPU.

the Spec-Ln-B2 and VL-Bn models are not significant, and some VL-Bn models show a slightly better win rate than SpecMCTS. As aforementioned, when a longer decision time is allowed, VL-Bn configurations also overcome the quality degradation through a large number of tree traversals.

VIII. CONCLUSION

The sequential nature of the MCTS may leave the abundant computing resources idle. SpecMCTS accelerates the MCTS process by utilizing a pair of DNN models. The two DNN models collaborate to accelerate the search process while reducing the impact on decision quality. The speculation model guides the MCTS agent to find important nodes to evaluate while the accurate node evaluations are performed in the background. Compared to the previous state-of-the-art, SpecMCTS achieves similar speed-ups while having better win rates when applied to the game of Go. SpecMCTS is most effective when each player is limited to a short amount of time to decide.

The performance and accuracy trade-offs in the speculation model have a significant influence on the decision quality of SpecMCTS. Future work can improve SpecMCTS by thoroughly evaluating a wide variety of model compression techniques to construct the speculation model. Also, the effectiveness of SpecMCTS in various domains should be investigated.

ACKNOWLEDGMENT

(Juhwan Kim and Byeongmin Kang contributed equally to this work.)

REFERENCES

[1] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang, "Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning," in *Proc. NIPS*, 2014, pp. 3338–3346.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and S. Dieleman, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and L. Zitnick, "ELF OpenGo: An analysis and open reimplementation of AlphaZero," in *Proc. ICML*, 2019, pp. 6244–6253.

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Euro. Conf. Mach. Learn.*, 2006.

[6] G. M.-B. Chaslot, M. H. Winands, and H. J. V. D. Herik, "Parallel Monte-Carlo tree search," in *Proc. Int. Conf. Comput. Games*, 2008, pp. 60–71.

[7] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[8] A. Liu, J. Chen, M. Yu, Y. Zhai, X. Zhou, and J. Liu, "Watch the unobserved: A simple approach to parallelizing Monte Carlo tree search," in *Proc. ICLR*, 2020.

[9] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 220–233.

[10] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *Proc. ISCA*, 2020, pp. 940–953.

[11] C. D. Rosin, "Multi-armed bandits with episode context," *Ann. Math. Artif. Intell.*, vol. 61, no. 3, pp. 203–230, 2011.

[12] Y. Soejima, A. Kishimoto, and O. Watanabe, "Evaluating root parallelization in go," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 278–287, Dec. 2010.

[13] L.-C. Lan, W. Li, T.-H. Wei, and I.-C. Wu, "Multiple policy value Monte Carlo tree search," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 1–7.

[14] L.-C. Lan, M.-Y. Tsai, T.-R. Wu, I.-C. Wu, and C.-J. Hsieh, "Learning to stop: Dynamic simulation monte-carlo tree search," 2020, *arXiv:2012.07910*. [Online]. Available: https://arxiv.org/abs/2012.07910

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. NIPS*, 2015, pp. 1–9.

[16] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[17] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[18] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NIPS Deep Learn. Workshop*, 2015.

[19] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 1–7.

[20] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2016, *arXiv:11608.08710*. [Online]. Available: https://arxiv.org/abs/11608.08710

[21] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. ICLR*, 2016, pp. 1–14.

[22] M. A. Carreira-Perpinan and Y. Idelbayev, "'Learning-compression' algorithms for neural net pruning," in *Proc. CVPR*, Jun. 2018, pp. 8532–8541.

[23] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "HRank: Filter pruning using high-rank feature map," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1529–1538.

[24] *NVIDIA TensorRT: Programmable Inference Accelerator*, NVIDIA, Santa Clara, CA, USA, 2021.

[25] B. Lee, A. Jackson, T. Madams, S. Troisi, and D. Jones, "Minigo: A case study in reproducing reinforcement learning research," in *Proc. ICLR*, 2019.

JUHWAN KIM received the B.S. degree in computer science and engineering from Sungkyunkwan University (SKKU), South Korea, in 2021, where he is currently pursuing the M.S. degree in computer science and engineering. His research interests include AI accelerator architectures and hardware design.

BYEONGMIN KANG received the B.S. degree in information and communication engineering from Myungji University, South Korea, in 2017. He is currently pursuing the M.S. degree in computer science and engineering with Sungkyunkwan University (SKKU), South Korea. His research interest includes accelerator architectures in FPGA.

HYUNGMIN CHO received the B.S. degree in computer science engineering from Seoul National University, South Korea, in 2005, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, in 2010 and 2015, respectively. He was a Research Scientist at Intel Labs, Santa Clara, CA, USA. From March 2017 to August 2019, he was an Assistant Professor with Hongik University, South Korea. He is currently an Assistant Professor in the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), South Korea. His research interests include reliable computer systems and accelerator architectures for high-performance computing.