Scaling the Inference of Digital Pathology Deep Learning Models Using CPU-Based High-Performance Computing

Weizhe Li, Mike Mikailov, *Member, IEEE*, and Weijie Chen

***Abstract*—Digital pathology whole-slide images (WSIs) are large-size gigapixel images, and image analysis based on deep learning artificial intelligence technology often involves pixelwise testing of a trained deep learning neural network (DLNN) on hundreds of WSI images, which is time-consuming. We take advantage of high-performance computing (HPC) facilities to parallelize this procedure into multiple independent (and hence delightfully parallel) tasks. However, traditional software parallelization techniques and regular file formats can have significant scaling problems on HPC clusters. In this work, a useful computational strategy is designed to localize and extract relevant patches in WSI files and group them in Hierarchical Data Format version 5 files well suited for parallel I/O. HPC's array job facilities are adapted for hierarchical scaling and parallelization of WSI preprocessing and testing of trained algorithms. Applying these techniques to testing a trained DLNN on the CAMELYON datasets with 399 WSIs reduced the theoretical processing time of 18 years on a single central processing unit (CPU) or 30 days on a single graphics processing unit to less than 45 h on an HPC cluster of 4000 CPU cores. The efficiency–accuracy tradeoff we demonstrated on this dataset further reinforced the importance of efficient computation techniques, without which accuracy may be sacrificed. The framework developed here for testing DLNNs does not rely on any specific neural network architecture and HPC cluster setup and can be utilized for any large-scale image processing and big-data analysis.**

***Impact Statement*—The exponential growth of imaging and other biomedical data along with advanced AI technologies hold greart promise to revolutionize medicine. However, handling big data and intensive computations represent a tremendous challenge. Our techniques of data handling and computations parallelization provide a seamless and efficient solution for imaging and other big-data applications. Specifically, we demonstrated the usefulness of our parallelization technique coupled with the HDF5 file handling for processing large amount of image data in a high-performance computing environment. Such a technique is particularly useful when an HPC facility is readily available and GPU resources are limited. Moreover, when a computation task involves code that has been programmed only for CPU, the CPU code can limit the use rate of GPU thereby becoming the bottleneck of the computational job. Although one could run the two parts in an HPC facility and a GPU facility separately, this would need extra storage space for saving data and extra overhead time for file I/O and data transfer. Our technique as a seamless and efficient solution is generally useful for big-data and computationally intensive applications.**

***Index Terms*—Artificial intelligence (AI), distributed architectures, machine learning (ML), parallelism and concurrency.**

I. Introduction

DIGITAL pathology has become popular thanks to technological advances in whole-slide image (WSI) scanners, image viewers, and displays, as well as the regulatory approval of such technologies by the US Food and Drug Administration (FDA) for primary diagnosis. The advantages of WSI-based digital pathology over traditional microscopy methods include remote consultation and diagnosis, ease of access to archival cases, and computerized image analysis. The availability of the big amount of WSI data has spurred large-scale data analytic applications, giving birth to the field of computational pathology, where artificial intelligence (AI) and machine learning (ML) algorithms are developed for a variety of clinical tasks, such as cancer metastasis detection, cancer subtype classification, mitotic cell counting, and tumor-infiltrating lymphocyte quantification, among many others.

One of the mainstream choices of AI/ML algorithms in digital pathology WSI applications is deep learning neural networks (DLNN) [1]. As is well known, DLNN involves estimating millions of parameters and is computationally intensive. Furthermore, DLNN presents unique challenges in digital pathology applications because WSIs are multiple-resolution gigapixel microscopic images that are much larger than images in other computer vision applications (e.g., natural images and radiological images). The graphics processing unit (GPU) technology is a natural choice for these highly complex big-data applications, due to their parallel nature. The central processing unit (CPU)-based DLNN training parallelization solutions usually involve significant modifications of the current training framework [2], which limits the usage of CPU on training popular DLNN architectures. Although DLNN training typically uses GPU, other time-consuming tasks, such as image preprocessing and testing of a trained DLNN model on a test dataset of hundreds or even thousands of WSIs, can utilize CPU-based distributed computation facilities to speedup the process. Therefore, the use of available high-performance computing (HPC) clusters in such

Manuscript received 7 March 2022; revised 7 November 2022; accepted 11 February 2023. Date of publication 17 February 2023; date of current version 22 November 2023. This article was recommended for publication by Associate Editor Mehmet Onder Efe upon evaluation of the reviewers' comments. *(All authors contributed equally to this work.) (Corresponding authors: Mike Mikailov; Weijie Chen.)*

Weizhe Li was with the U.S. Food and Drug Administration, Silver Spring, MD 20993 USA. He is now with Pangiam, McLean, VA 22102 USA (e-mail: weizheli@gmail.com).

Mike Mikailov and Weijie Chen are with the U.S. Food and Drug Administration, Silver Spring, MD 20993 USA (e-mail: mike.mikailov@fda.hhs.gov; weijie.chen@fda.hhs.gov).

Digital Object Identifier 10.1109/TAI.2023.3246032

CPU-friendly tasks can greatly facilitate performance assessment studies of AI algorithms for digital pathology applications, especially when the GPU facility is limited. Our purpose in this article is to present computational techniques for scaling digital pathology deep learning AI at the stage of inference (testing) on CPU-based HPC clusters, which are easy to implement and independent of DLNN architectures and HPC cluster setup.

WSI image analysis (e.g., lesion classification and detection) often involves pixelwise classification, which has also been widely used for general image segmentation, scene labeling, and object detection and tracking, where the output of the analysis is a heatmap, with each pixel value representing the likelihood that certain signal or condition of interest exists [3], [4], [5], [6]. The widely used deep learning classifier, namely the convolutional neural network (CNN), is usually trained using small image patches with binary labels (with versus without the condition). When the trained CNN is applied to a test WSI, the score for one or a few pixels is determined by the trained neural network using the subimage centered at the pixel(s) as input, i.e., a sliding window of the same size as the training patches. The sliding window method was the first method introduced to produce pixelwise classification and is still widely used because it does not rely on specific architectures of neural networks [7]. However, the sliding window method is computationally expensive on gigapixel WSIs. Although GPU has been routinely adopted for its high capacity of dealing with computationally intensive tasks by massive parallelization, the efficiency of a single GPU is limited given the gigapixel WSI images [7] due to the large number of patches it would have to process. Alternative solutions, such as parallelization techniques, in a CPU-based HPC environment are highly motivated for DLNN in digital pathology.

One of the most challenging tasks in using HPC clusters and GPU platforms for solving increasingly computation- and data-intensive deep learning algorithms is how to parallelize and scale these algorithms to take advantage of massively parallel hardware resources efficiently. Traditional software parallelization techniques, such as OpenMP and POSIX multithreading, are designed for shared-memory computing environments confined to one computing node and therefore cannot be applied to distributed computing environments, like HPC clusters. Limitations of the most dominant parallelization technique Message Passing Interface (MPI), used in distributed environments, are described in Section II.

Another challenge in the parallel processing of WSIs is their extremely large sizes. A single WSI in uncompressed format could be up to 30–40 GB in size, which creates challenges in loading the entire image to the memory of a computing node for processing. To overcome these challenges, traditional approaches introduce a preprocessing step, where the images are split into smaller size patches and stored as separate files in the storage system [8]. Applying this technique to WSI datasets (like in CAMELYON) used for deep learning algorithms may produce millions of such patch files, which leads to contention and I/O failures when too many files are to be opened at the same time. Thus, this approach limits the scalability of the application by placing a heavy burden on the storage systems and computational infrastructure of the HPC clusters.

To overcome these deficiencies, another novel aspect of our approach is to extract patches from WSIs, group them, and save them in Hierarchical Data Format version 5 (HDF5) [9] files as a preprocessing step. The HDF5 format is well suited for parallel data access (PDA) via Lustre [10], IBM Spectrum Scale also known as general parallel file system (GPFS) [11], or any other parallel cluster file system. For each WSI, only one HDF5 file is created, which contains groups of patches extracted from the WSI. A Python package py_wsi [12] is available for converting and saving a single WSI file in an HDF5 file to facilitate deep learning with WSI. However, this package does not split the WSI into sets of patches and strategically group them in the HDF5 file to help with parallel processing. In this work, we further developed the technique by taking advantage of the built-in hierarchical features of HDF5 for grouping WSI patches to facilitate their parallel processing. At the WSI processing step, each group in the HDF5 file is accessed and processed by one array job task in a distributed, parallel, and scalable manner.

We demonstrate our computational technique using a deep learning pipeline on the WSIs from CAMELYON datasets. Specifically, part of the pipeline using our technique includes the following.

1) *Data preparation stage, also referred to as patch extraction and grouping step.* At this stage, the WSIs are split into smaller size patches, grouped, and then saved in HDF5 files in parallel using the array job technique of the job scheduler on our HPC cluster.
2) *Heatmap generation stage.* The patch groups created at the data preparation stage are processed to generate heatmaps through a trained DLNN using the hierarchical scaling technique proposed in this article.

The rest of this article is organized as follows. Section II describes the existing methods and their limitations for processing large-scale WSIs on high-performance computing platforms. Section III presents our novel techniques and implementation for the parallel processing pipeline. Section IV contains the results of our experiments in applying the techniques to WSIs from CAMELYON datasets. Section V presents the general applicability of the techniques presented in this article. Finally, Section VI concludes this article.

II. Existing Methods and Their Limitations

A. GPU-Based Techniques

The dominant use of GPU is because of its large numbers of processors and the easy access to CUDA libraries such as the NVIDIA CUDA Deep Neural Network (cuDNN) by high-level programming languages, such as C/C++ and Python, which make GPU easier to program in a wide range of computational tasks [13]. GPU is generally used for DLNN training involving intensive matrix multiplications in the standard back-propagation training framework. Data and model parallelization are the major methods used for GPU-based training parallelization, which suffers from scaling and efficiency issues [14], [15]. For the testing of a trained neural network, several methods have been developed to speedup pixelwise classification on GPU platforms. The "sparse/strided kernels" methods avoid lots of

redundancy of the sliding-window method by introducing *d*-regularly sparse kernels, which are formed by inserting multiple zeros in the columns and rows of regular convolutional and pooling kernels and separating the entries of the original kernels *d* pixels away. The combination of sparse kernels and 1-strides allows continuous access to GPU memory and maximization of GPU use [16], [17]. However, the computation for strided kernel convolution was not supported by cuDNN when it was published.

Fully convolutional networks also provide an efficient way for pixel-level prediction of an image by adding upsampling transposed convolution layers in the network [18], [19], [20]. After a series of transposed convolution layers, the output of the prediction is usually a score map ("labelmap") of the same size as the input image. The regular classification network relying on sliding windows for the prediction can produce fast pixelwise prediction by adding transposed convolution layers [16].

The convolutional implementation of the sliding window algorithm is now a common method to speedup pixelwise prediction using classification models. The sliding window algorithm is embedded by converting the fully connected layer to the convolutional layer after classification neural networks get trained [21]. The modified neural network is able to accept input images with the size larger than the training image, therefore increasing the prediction speed.

Although the above-mentioned GPU-based solutions can increase the prediction speed significantly, the traditional sliding window method is still widely used as it is independent of the architecture of classifiers. The "sparse/strided kernels" and fully convolutional networks require adopting these networks to make fast prediction possible. The method of converting the fully connected layer to a convolution layer requires a preset output dimension, which limits the choices of heatmap resolution, whereas the sliding window method allows more flexible resolution choices by setting the stride size (skipping pixels). Moreover, if the prediction involves CPU-based image processing, such as color normalization, the speedup effect will be mostly counteracted by the long processing time on the CPU, which bottlenecks GPU use. Recently, image compression and resolution reduction have been adopted to reduce the computation burden in gigapixel WSI and radiographic image applications [22], [23]; however, the fast training and inference were reported to tradeoff with the price of suboptimal classification performance. In this work, we propose an HPC-based scaling up method that is independent of the architecture of classifiers and can be used for speeding up sliding-window-based methods as reported here and other methods such as fully convolutional networks.

B. HPC-Based Approaches With Traditional Parallelization Techniques

The most dominant parallelization technique used in distributed environments is MPI [24]. However, current implementations of MPI require all needed resources to be available simultaneously for the jobs to start. This may lead to "job starvation": Jobs requesting significant resources may never be scheduled or may be delayed in the queue for a prohibitively long time, even if their priorities are high and increasing over time [25]. This is because available resources are first assigned to jobs requesting a small amount of resources. On the other hand, applying the reservation feature to reserve a large amount of resources may lead to low system performance.

Moreover, MPI does not include the checkpointing feature by default. As the number of CPU cores requested by a large-scale application increases, the probability of getting unhealthy computing nodes also increases, which in turn leads to unrecoverable failures.

Even though MPI supports dynamic processes, current HPC job schedulers assign only a fixed amount of resources (e.g., CPU cores) to MPI jobs at their start-up, and this number cannot exceed the current capacity of the clusters. Therefore, newly available resources of the clusters cannot be assigned (under the control of the job schedulers) dynamically to running MPI applications. This leads to two scalability limitations: 1) The amount of requested resources cannot exceed the maximum capacity of the clusters; 2) newly available resources cannot be assigned to running MPI applications under the control of the job schedulers.

The built-in array job facility of the job schedulers [26] used in this work for processing WSI in a perfectly parallel manner (no intertask dependencies exist) avoids the above-mentioned limitations. This facility can launch parallel independent tasks of an array job across the distributed computing environment of the HPC clusters in a scalable manner—the array job starts whenever computational resources are available, even for a single task of the job, thereby avoiding the job starvation problem. Other tasks of the job are started automatically by the job schedulers as more resources become available. The number of tasks in a single array job can exceed the maximum capacity of the HPC clusters; however, the number is limited by the system implementations. A hierarchical scaling technique based on the array job mechanism offered in this work overcomes the system limitation of the maximum number of tasks per array job as well.

Array jobs also provide natural checkpointing capabilities, and a system failure can thus affect only a subset of running tasks. Only the failed tasks need to be rerun to recover from the system failures. There are workflow systems, such as Swift [27] and Pegasus [28], designed to avoid MPI's rigidity and lack of elasticity. However, there are some limitations, which are listed as follows.

1) The workflows need to be learned, which could be a burden, whereas our approach is based on widely used and well-known open-source programming tools, Python and Linux.

2) None of these workflows provide an integrated approach (as presented in this work) for scaling digital pathology AI (DPAI) on HPC clusters.

3) The workflows do not address the system limitation of the job schedulers, whereas hierarchical scaling is introduced to avoid the limitation.

The use of HPC clusters and parallel processing algorithms for digital pathology deep learning AI is currently limited. Yildirim and Foran [8] investigated deficiencies in existing approaches [29], [30], [31], [32], [33] and proposed two enhancements: 1)

PDA, which makes use of parallel file systems, such as Lustre [10] and MPI-based libraries; and 2) distributed data access (DDA), which utilizes distributed file/object storage systems such as hadoop distributed file system (HDFS) and amazon web services simple storage service (AWS S3). Even though the PDA approach improves the scalability of WSI processing by distributing/delegating read operations to worker nodes (versus a single-master node in earlier designs), this approach 1) inherits (as other MPI-based technologies) the limitations of the MPI implementations mentioned earlier in this article and 2) adds its own deficiency by allowing potentially thousands of worker nodes to access the same WSI file saved in its original format (not well suited for parallel access), which may lead to unrecoverable system failures [34]. Thus, this approach limits the scalability of the application by making the deployment of a trained network more vulnerable to I/O errors. The DDA approach, on the other hand, dynamically distributes WSIs to YARN containers (no parallel cluster file system is needed) and concentrates the processing of a whole image to only one YARN container (computing node), which in turn limits the scalability of the image processing. The hierarchical scaling technique introduced in this work distributes subsets of a single WSI (extracted and grouped in an HDF5 file beforehand) to different computing nodes, thus avoiding scalability limitation.

A modular pipeline with three independent layers for detection of tumor regions in digital specimens of breast lymph nodes with deep learning models using CAMELYON dataset is described in [35]. Similar to our work, this pipeline also uses CPU-based HPC for image preprocessing and patch extraction, along with HDF5 as an intermediate database.

1) However, the work presented in [35] showed that 10 000 CPU cores were used to extract data from 5 WSIs in about 1 h. Using our techniques, patches can be extracted from 399 WSIs and grouped in HDF5 files in less than 3 h using only 399 CPU cores.
2) In [35], the patches were not grouped in HDF5 files. GPUs were used, which are scarce and do not scale well because of the limited amount of RAM accompanied by the massive compute capacity [36]. The use of HDF5 files in our techniques facilitates distributed and parallel processing on the CPU-based HPC at the inference step.

III. Materials and Methods

A. CAMELYON Dataset

The CAMELYON16 challenge aimed to assess deep learning algorithms in detecting metastases in hematoxylin- and eosin-stained tissue sections of the lymph nodes of women with breast cancer. The dataset contains 399 WSIs: 270 training WSIs (159 normal WSIs and 111 tumor WSIs) and 129 testing WSIs. These WSIs were acquired from two medical centers, the Radboud University Medical Center and the Utrecht University Medical Center, where the WSIs were acquired at a resolution of 0.23–0.24 μm per pixel using a 3D Histech (Budapest, Hungary) Pannoramic 250 Flash II digital slide scanner and a Hamamatsu (Hamamatsu, Japan) NanoZoomer-XR C12000-01 digital slide scanner, respectively [37], [38]. The acquired WSI images were then transformed into bigTiff format from their generic formats with JPEG compression and were stored as a pyramid of images composed of several levels of magnifications [37].

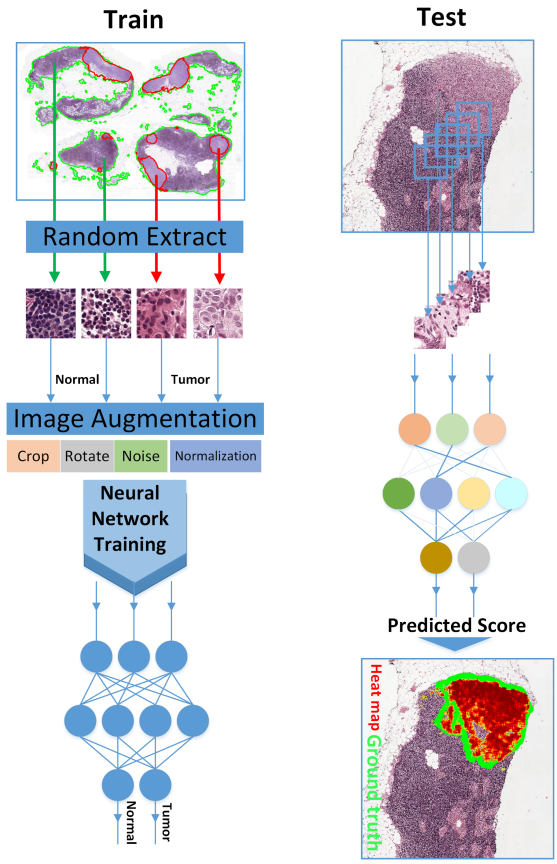

Fig. 1. DLNN pipeline for digital pathology. In training, the normal and tumor image patches were randomly extracted from normal (green contours) and tumor (red contours) tissue regions. After image augmentation (e.g., random cropping, rotation, adding color noise, and color normalization), the image patches were used to train a neural network on GPUs (Inception v1 in this work). In testing, the trained classifier is used to generate a pixelwise heatmap (red) for lesion detection and classification on WSI images by a sliding window (blue box), which is time-consuming, and a CPU-cluster-based parallelization framework is proposed in this study to significantly speedup this step.

The ground truth includes the slide-level binary labels (i.e., whether the slide contains tumor or is normal) and XML files delineating the contours of metastatic cancer regions on tumor slides. The ground truth was established by two pathologists, who utilized immunohistochemistry (anti-cytokeratin) when they could not agree with each other [37].

B. Deep Learning Pipeline

Fig. 1 illustrates part of our processing pipeline for the CAMELYON detection/classification problem. Although a trained neural network was used in this study, here we still list the key parameters and steps for model training. Because the tissue region only accounts for approximately 20% of the whole slide [39], we first segmented the tissue regions from the WSI image to focus the subsequent analysis on these regions. For tissue segmentation, the WSIs were first downsampled to fit into RAM by 32 folds over their length and width. The

downsampled image was then converted from red, green, blue (RGB) color space to HSV color space, and optimal threshold values for Hue (H), Saturation (S), and Value (V) channels were retrieved by the Ostu algorithm [40]. The threshold values were applied to the downsampled image to generate a binary mask and, subsequently, a bounding box for the tissue region. The image patches with a size of 256 × 256 pixels were randomly extracted from the region enclosed in the bounding box and screened by the threshold values to exclude the image patches from the background area.

For training WSIs, truth mask images with the same size as WSIs were generated from XML files containing ground truth information provided by CAMELYON using the Automated Slide Analysis Platform [37]. The training image patches of size 224 × 224 were generated from random cropping of the 256 × 256 patches. The tumor image patches were extracted and labeled as "1" if at least 50% of the pixels in the corresponding truth mask image patch are tumor pixels; the normal image patches were extracted and labeled as "0" if none of the pixels is a tumor pixel. All the training image patches were extracted using OpenSlide [41]. The extracted training image patches were then augmented with random cropping, horizontal flipping, rotation (90°, 180°, 270°), color normalization [42], and adding color noise (The image patches were converted from RGB to HSV color space, and a random integer from 0 to 20 was added to the H, S, and V channels. Then, image patches were converted back to the RGB color space.) [43]. We used these training patches to train the Inception V1 network [44] on a GPU platform.

Note that the input to the neural network is a patch image (of size 224 × 224, in our case) and the output is a single score representing the probability of a tumor. In testing, each pixel can have such a score output using a patch image centered at that pixel as input (i.e., a sliding window) to the neural network, thereby resulting in a pixelwise heatmap. This is a computationally intensive procedure due to the large number of pixels in a WSI. Furthermore, we applied the heatmap generation procedure to all the WSIs, including both the training and test images. A supervised classifier (random forest, RF) was trained for slide-level classification in our pipeline (not shown in Fig. 1; refer to [38] for more details). The feature extraction and RF model training were conducted based on the descriptions given in [38] and [44]. Therefore, the time-consuming pixelwise heatmap generation (a.k.a. prediction) procedure has to be applied to all the 399 WSIs in the CAMELYON16 dataset. Fortunately, it is a natural parallel task that can take advantage of HPC capabilities. We used the HPC cluster at the FDA for this task. Next, we describe the techniques we used in this facility.

C. Patch Extraction and Grouping in the HDF5 File

To generate a heatmap for each WSI, we first extract patches from the tissue regions and put them in an HDF5 file, which are retrieved later for prediction. As shown in Fig. 2, each WSI is divided into a matrix composed of 224 × 224 image patches. Each patch is determined as "tissue" (coded as 1) or "background" (coded as 0) using the tissue thresholds by the Ostu algorithm as described in the previous section, but only applying to the H and S channels to make sure most of the

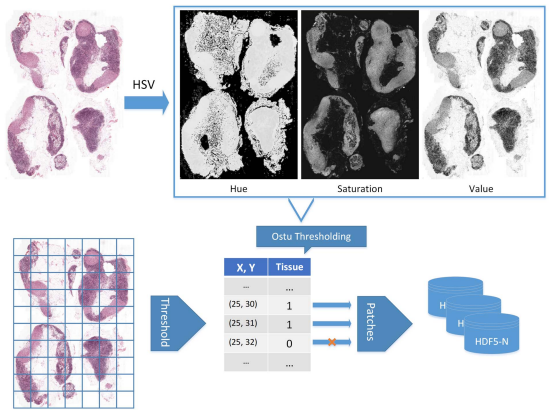

Fig. 2. WSI image segmentation and look-up table construction. The (*X*, *Y*) coordinates are indices of a patch in the patch-matrix, counting from the upper-left corner of the WSI.

tissue regions are preserved for testing (thresholding all the three channels may lose some tissue regions). The indices in the patch matrix and the tissue/background codes of these patches are stored in a table such that only tissue patches are processed for heatmap generation, and these heatmap patches can be correctly stitched together or overlapped on the WSI. It is important to note, however, that each patch actually extracted and stored in the HDF5 file for processing is of size 448 × 448, which is centered at each target 224 × 224 patch and extended 112 pixels on each of the four sides. This is because, as mentioned earlier, the prediction for each pixel in the target 224 × 224 patch requires an input patch of size 224 × 224 to the neural network and the extension is to have the input patch for every pixel in the target 224 × 224 patch.

An HDF5 file contains a root group that, in turn, contains other groups. A group is a folder-like structure that may contain other groups or datasets within it. The datasets may contain many different types of data. HDF5 supports concurrent read access to datasets in a single HDF5 file from multiple processes running on the same or different computing nodes across the HPC cluster.

Patches of size 448 × 448 pixels from 40x WSIs are extracted and grouped in HDF5 files [45]. A much larger size of image patches could be used here to further reduce the I/O burden because of the greater accessibility of larger memory on the CPU compared to the limited memory of the GPU. The image size of 448 × 448 pixels is used here to avoid nontissue regions as much as possible. The same image size and batch size are also used for testing the trained model on GPU to ensure the loaded images fit the GPU memory. One HDF5 file containing groups of up to 400 patches (a total of up to ∼230 MiB) is produced for each slide. As mentioned earlier, only the patches from the tissue region are included in the groups, which significantly decreases the number of groups, thereby reducing processing time and the required storage size.

Each WSI image is converted into one HDF5 file. The number of groups in each HDF5 file is variable and depends on the size of the WSI image and the number of patches in a group. Recall that "group" is the unit of data assigned to and processed by one array job task. More than 80% of computing nodes in the

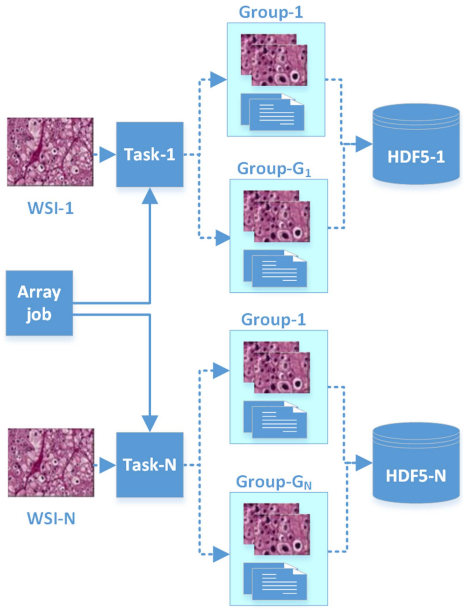

Fig. 3. Extracting and grouping of WSI patches in HDF5 files.

HPC cluster contain eight CPU cores. By using only seven CPU cores for processing a group, we reserve one CPU core for the operating system's needs. With one patch size of 448 × 448 × 3 bytes, the maximum size of a group (with 400 patches) equals ∼230 MiB. The number of patches (N_p) per group could be determined by the formula $N_p = \lfloor M/S \rfloor$, where M is the amount of available RAM for the application in a computing node, S is the size of one patch, and $\lfloor \cdot \rfloor$ denotes the floor integer. Then, the number of groups (N_g) in an HDF5 file could be determined by $N_g = \lceil T_p/N_p \rceil$, or $N_g = \lceil T_p/(\lfloor M/S \rfloor) \rceil$, where T_p is the total number of patches of size S in one WSI that has been determined in patch extraction and $\lceil \cdot \rceil$ denotes the ceiling integer. For example, for a randomly selected WSI in our datasets, we ended up with 22 199 tissue patches. Considering that we have a total of 4000 CPU cores available at a time for our use and 7 CPU cores are used for processing a group using the multithreaded Python program, we decided to put 400 patches (assuming 230 MiB of RAM is available for the group data) in each group, and so this WSI ended up with 56 groups [56 = CEILING (22199/4001)] in the HDF5 file. With these settings, the total processing time of all groups in all 399 HDF5 files is less than 45 h. Every group in an HDF5 file is also accompanied by WSI metadata containing information about the patches (e.g., coordinates of the patches in the WSI) in the group. The built-in array job facility of Son of Grid Engine (SGE) [46] is used to launch N independent tasks to parallelize and scale reading patches from each slide, generating groups of patches, and placing them in HDF5 files as shown in Fig. 3. Here, N is the number of slides, and each slide is processed by one independent task in parallel.

LISTING 1 in the Appendix contains a Linux shell script code excerpt that creates a list of names (in file LISTFILE) of WSIs located at directory FILE_DIR and submits an SGE array job using the ARRAY_SCRIPT script, which in turn uses LISTFILE to launch one task per WSI for extracting patches from the WSI, grouping them, and saving them in an HDF5 file.

The ARRAY_SCRIPT SGE script code excerpt is shown in LISTING 2 (see Appendix). Every task of this array job uses the Python program split_grp.py (source code could be found in our GitHub repository [47]) for extraction, grouping, and saving patches in HDF5 files.

After completion of all the array job tasks, a special procedure (pseudocode shown in LISTING 3 in the Appendix) is run to create a look-up table associating (in each row) an HDF5 file with a pair of numbers indicating the first and last group IDs in the HDF5 file. The look-up table is used later for dynamically launching array jobs for processing the groups in all HDF5 files: The number of array jobs is determined by the number of rows in the table, and the number of tasks in each array job is determined by the corresponding pair of group IDs.

D. Hierarchical Scaling

The number of groups of the image patches produced at the data preparation stage may well exceed the maximum number of tasks allowed within an array job, which in turn limits the scalability of the image processing using an array job. Instead, a hierarchical approach is taken in which the master job launches N array jobs, each of which in turn runs Gi tasks, where N is the number of WSIs and Gi is the number of groups in the ith WSI as shown in Fig. 4. Once launched, every task retrieves patches from the respective group using a Python program, OpenSlide, and HDF5 libraries and processes them. Seven CPU cores are used in a task for processing the respective group.

The master job code excerpt is shown in LISTING 4 in the Appendix. This job uses the look-up table prepared earlier to determine (for each WSI) the first and last task IDs of array jobs associated with the WSI and its corresponding HDF5 file. Then, it creates working directories and runs the array jobs with a varying number (corresponding to the number of groups in each HDF5 file) of Gi tasks using an SGE script process_array.sh.

A code excerpt from the process_array.sh SGE script is shown in LISTING 5 in the Appendix. Based on its task ID, SGE_TASK_ID, every task of this array job script determines the group number in the HDF5 file for the associated WSI and runs a Python program (source code could be found at our GitHub repository [47]), which in turn uses OpenSlide and HDF5 libraries to retrieve and process the patches of WSIs. The testing scores of pixelwise classification for extracted image patches in each group of HDF5 files are generated using the trained neural network and stored in the format of a Python numpy array with hierarchical labels of HDF5 file name and group name. These labels are then used to correlate the testing scores (numpy arrays) to the respective image patches and the relative positions in WSI images. Then, heatmaps were constructed by stitching the numpy arrays according to their positions (see Fig. 4).

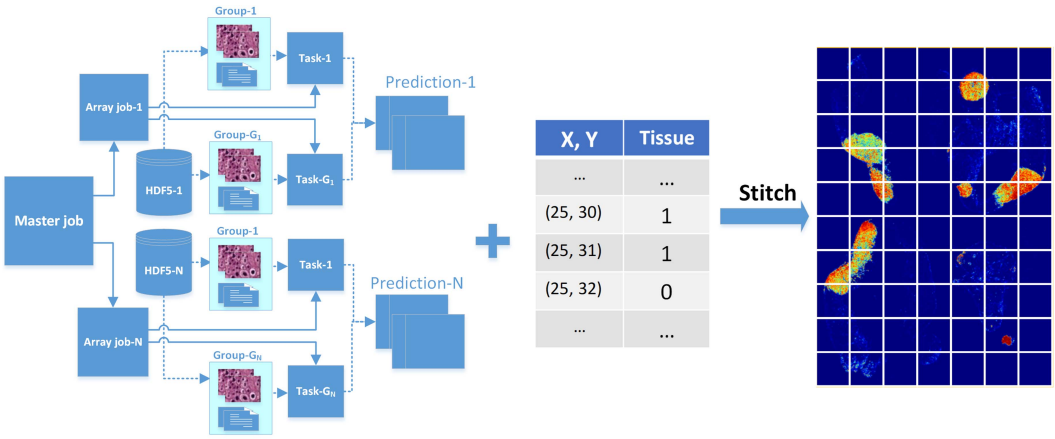

Fig. 4. Hierarchical scaling technique and heatmap construction.

IV. Experimental Results

In this section, we present experimental evidence to demonstrate the computation efficiency of our techniques using run-time statistics (see Section IV-A). In addition, we investigated the tradeoff between computation efficiency and classification/detection accuracy, i.e., the effect of algorithmic choices for computation efficiency on accuracy (see Section IV-B).

A. Computation Efficiency

The techniques presented in this work have been applied to deep learning AI using 399 WSIs from CAMELYON 16 datasets [48]. As noted in Section III-B, this includes both the training (159 normal WSIs and 111 tumor WSIs) and the testing (129 WSIs) subsets. We tested two trained DLNNs: One that requires applying a color normalization procedure to the patches before testing, and one that does not use such preprocessing. The prediction and heatmap generation in the deep learning pipeline were run on our Betsy HPC cluster of the U.S. Food and Drug Administration. This cluster contains a total of 4408 cores (x86_64 architecture, Intel Nehalem) and 40 TB of RAM in 405 computing nodes interconnected with 40 Gb InfiniBand and 10 Gb Ethernet communication paths. All computing nodes run CentOS Linux release 7.3.1611. SGE is used as a job scheduler, and GPFS [11] is used as a parallel cluster file system.

Table I contains run-time statistics for the extraction and grouping of WSI patches in HDF5 files. All 399 array job tasks have been run in parallel, and the longest task has taken 2.68 h, which led to a total of 132-fold estimated speedup.

Table II contains run-time statistics for the prediction (i.e., heatmap) generation step. Since seven CPU cores are used for a group, a total of 190 960 CPU cores are needed to process a total of 27 280 groups placed in 399 HDF5 files beforehand. Applying the hierarchical scaling technique on the Betsy cluster, the needed computational resources have been provided in batches of 4000 CPU cores seamlessly achieving estimated total speedups of 3784.89 (95% parallel efficiency) and 3947.41 (99% parallel efficiency) for the DLNN models without and with color normalization procedures, respectively.

TABLE I
Run Time Statistics for Extraction and Grouping in HDF Files

	WSI subsets			Total
	Training(normal)	*Training (tumor)*	*Testing*	
Number of slides	159	111	129	399
Size, (GB)	277	218	206	701
Extraction and Grouping				
CPU cores	159	111	129	399
Estimated serial run time,[1] S_1 (hour)	173	91	90	354
Parallel run time, P_1 (hour)	2.68	2.16	2.50	2.68*
Estimated speedup (S_1/P_1)	65	42	36	132§
Number of groups created	10 464	9 452	7 364	27 280
Size of HDF5 files, (GB)	2329	2108	1639	6076

*Because this is parallel run time, the total is the longest running time, not the sum of the run times for the three subsets.
§Because the three WSI subsets shown in the table were running in parallel, the speedups are calculated separately. The "total" speedup is calculated as the ratio of the total serial run time (354) over the total parallel run time (2.68).
[1]Sum of run times for processing each slide in every WSI subset.

Thanks to grouping image patches in HDF5 files, we achieved a significant amount of I/O reduction: about 400 times fewer accesses to the file system. There are 400 patches in one group, and every task reads the whole group (∼230 MiB) at once. Further experiments showed that as the number of patches in a group increases from 400 to 1600 (919 MiB), the reading speedup of the whole group at once (compared to reading all the patches in the group individually) increases almost ten-fold, from 29 to 284, see Fig. 5, which in turn leads to a reduction of the total time needed for the whole pipeline execution. As the number of patches in a group exceeds 1600, the speedup decreases. Many factors (e.g., network bandwidth, operating system settings, file system configurations, system load, and available RAM on computing nodes) in distributed computing

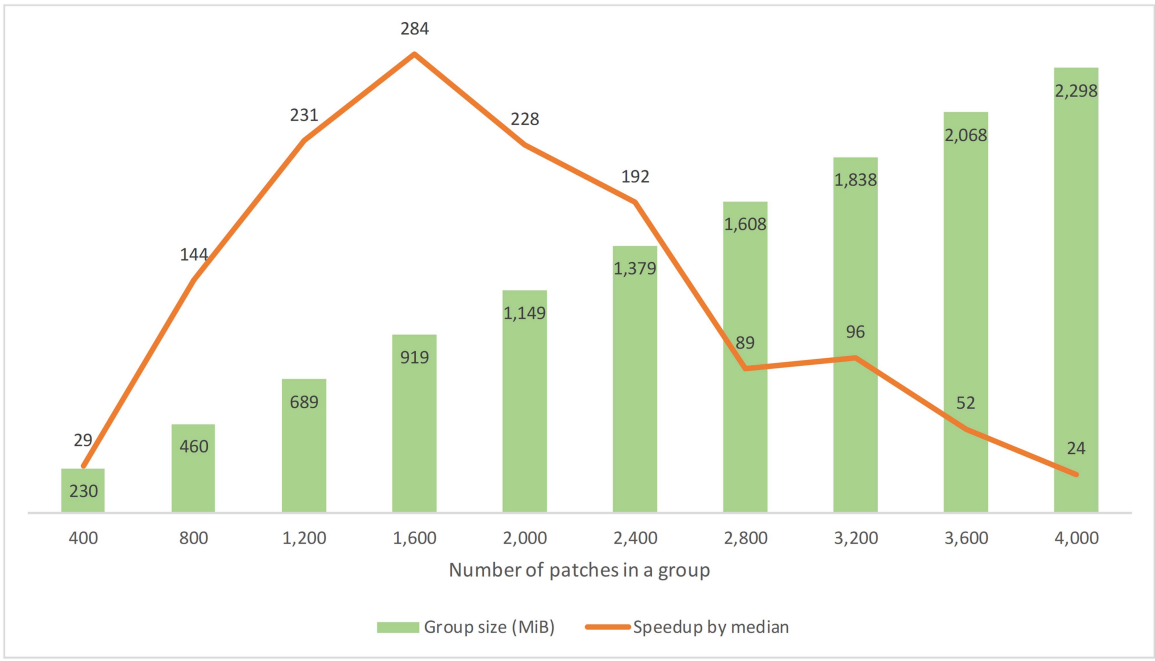

Fig. 5. Experimental results on group file reading speedup compared to reading all patches of the group individually. *X*-axis in the chart represents the numbers of patches in the groups, the red line graph represents group file reading speedup, and the bar graph reflects the sizes (in MiB) of the groups.

environments, such as HPC clusters, affect the performance of the I/O subsystem, and the peak of speedup shown in Fig. 5 reflects the tradeoff among these factors. A special tool is designed in Python to assess the speedup in various group configurations on an HPC cluster and find the most efficient configuration for a pipeline. As shown in Table III, for combined run time statistics, the total estimated serial run times of 15.23 and 18.55 years are reduced to an estimated 37.83 and 43.77 h without and with color normalization procedures, respectively. Estimated speedups are 3526 and 3714, with parallel efficiencies of 88% and 93%, respectively. As a comparison, it took 30 days to test the "without color normalization" model using a single GPU (NVIDIA GTX 2080 Ti) on the same WSI images in the same task.

B. Tradeoff Between Computation Efficiency and Classification/Detection Accuracy

Inference on WSI using a trained deep neural network is time-consuming due to the large image size. Algorithmic choices are frequently made to improve computation efficiency; however, the effect on accuracy is often unclear. One of the common practices in DLNN inference on WSI is to use a stride parameter to slide the DLNN input image window over the WSI for generating heatmaps (see Section II-A). A larger stride size decreases the inference time, but the gain in efficiency may come at a price in accuracy. Here, we present our experimental results investigating this effect.

We first investigated the effect of stride size on the slide-level classification (cancer versus noncancer) in terms of the area under the receiver operating characteristic (ROC) curve (AUC). Note that because the inference computation is time-consuming, we generated the heatmap with a stride size of 16 and then downsampled it to generate heatmaps that correspond to stride sizes of 64, 128, and 224. The same global and local features were extracted based on these heatmaps. RF models were trained separately corresponding to each stride size based on the features extracted from these heatmaps. The performance of the trained RF models was then assessed by ROC analysis. As shown in Table IV and Fig. 6, the RF models appear to have no significant difference in the slide-level classification performance. This indicates that heatmaps with lower resolution may be used for fast slide-level prediction when computational resources are limited. To understand this, we checked the contributions of individual features of the RF model; the description of these features can be found in [38]. We found that the features carrying the most weights are "largest area" and "maximum probability of the largest area," which are not sensitive to stride size. This explains the almost similar slide-level classification performance of the RF model across different stride sizes.

We then investigated the effect of stride size on the lesion-level detection performance. We used the free-response operating characteristic (FROC) curve to evaluate this performance, following the method in the CAMELYON16 challenge [38]. The FROC curve plots the sensitivity of the DLNN model in detecting metastases annotated by pathologists (truth) against the number of false-positive detections per WSI. Our results show that the stride parameter (i.e., heatmap resolution) has a substantial impact on the lesion detection performance (see FROC curves in Fig. 7). Table V shows the sensitivity values (i.e., true positive fractions) at various numbers of false positives (FPs) and their average (i.e., FROC score). It can be seen that the sensitivity decreases from 0.45 to 0.236 at 0.25 average FP when the resolution of the heatmap is reduced by the stride size of 16 to the stride size of 224. A possible reason for the low sensitivity

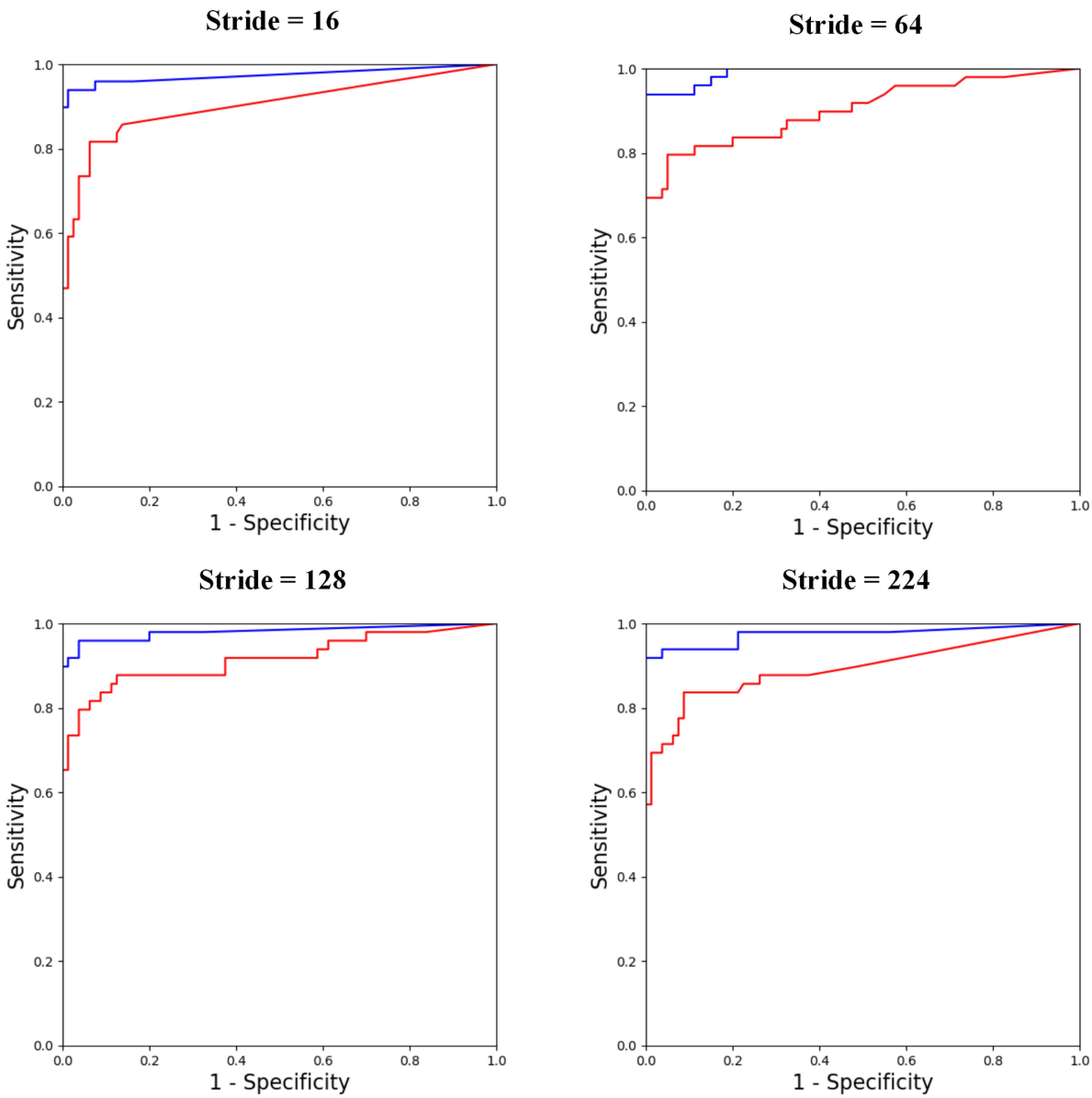

Fig. 6. Slide-level detection based on heatmaps corresponding to different stride sizes (red and blue correspond to without and with color normalization and color augmentation, respectively).

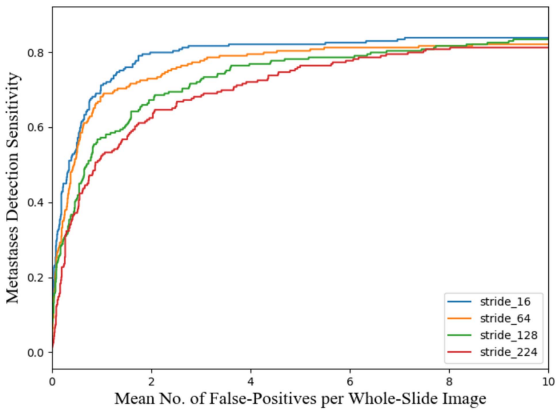

Fig. 7. FROC curves for the comparison of lesion-level detection performance over different heatmap resolutions.

at the left end of the FROC curve could be that the resolution of the heatmap is not high enough to capture small lesions.

V. General Applicability

The techniques presented in this work are generally applicable to scaling large-scale image processing problems and a wide variety of applications in bioinformatics, modeling, and simulation. For example, we have applied our parallelization technique to next-generation sequencing data for alignment and search of biological sequences [49], drug–protein interaction data for investigating how 3100 active drug ingredients can be expected to interact at a molecular level with 10 000 proteins known to exist in the human body [50], and Markov Chain Monte Carlo simulations [51], [52].

TABLE II
RUN TIME STATISTICS FOR PREDICTION GENERATION

	WSI subsets			Total
	Training (normal)	*Training (tumor)*	*Testing*	
CPU cores[1]	73 248	66 164	51 548	190 960
Number of batches[2]	18	17	13	48
Without color normalization				
Estimated serial run time,[3] S_2 (hour)	50 873	46 291	35 870	133 034
Estimated run time, single GPU, G (hour)	287	200	233	720
Parallel run time, P_2 (hour)	13	12	9	34
Estimated speedup (S_2/P_2)	3778	3789	3790	3788
Estimated speedup (G/P_2)	21	16	25	20
With color normalization				
Estimated serial run time, S_3 (hour)	62 264	56 262	43 651	162 176
Parallel run time, P_3 (hour)	16	14	11	41
Estimated speedup (S_3/P_3)	3947	3953	3941	3947

[1]Seven CPU cores are used for each group.
[2]In every batch, 4000 CPU cores are used.
[3]Sum of run times for processing each group.

TABLE III
COMBINED RUN TIME STATISTICS

	WSI subsets			Total
	Training (normal)	*Training (tumor)*	*Testing*	
Without color normalization				
Estimated serial run time, S_4 (hour)	51 046	46 382	35 960	133 388
Parallel run time, P_4 (hour)	16	14	12	38
Estimated speedup (S_4/P_4)	3161	3225	3007	3526
With color normalization				
Estimated serial run time, S_5 (hour)	62 437	56 353	43 741	162 531
Parallel run time, P_5 (hour)	18	16	14	44
Estimated speedup (S_5/P_5)	3383	3437	3223	3714

TABLE IV
AUCS (WITH 95% CONFIDENCE INTERVALS) OF SLIDE-LEVEL CLASSIFICATION PERFORMANCE BASED ON HEATMAPS WITH A VARIETY OF STRIDE SIZES

Stride size	Without color normalization and augmentation	With color normalization and augmentation
16	0.902 (0.844, 0.96)	0.974 (0.942, 1)
64	0.902 (0.841, 0.964)	0.991 (0.98, 1)
128	0.913 (0.857, 0.975)	0.981 (0.953, 1)
224	0.893 (0.827, 0.96)	0.975 (0.941, 1)

TABLE V
FROC ASSESSMENT OF CANCER LESION DETECTION

Strides	True-positive fraction at the different false positive (FP) values						FROC Score
	1/4 FPs/ WSI	1/2 FPs/ WSI	1 FPs/ WSI	2 FPs/ WSI	4 FPs/ WSI	8 FPs/ WSI	
16	0.45	0.541	0.712	0.80	0.821	0.838	0.694
64	0.358	0.528	0.681	0.729	0.795	0.817	0.651
128	0.306	0.406	0.572	0.672	0.769	0.817	0.59
224	0.236	0.371	0.524	0.624	0.721	0.808	0.547

Fig. 8 illustrates the generic procedure of the HPC scalable workflow applicable to general big-data processing applications (on the left; including generic script code) and the corresponding specific application to the inference of digital pathology deep learning AI as presented in this article (on the right; including reference to the listing of script code in the Appendix). It should be noted that the techniques demonstrated in the DPAI application can be applied to any imaging AI application that involves image processing and testing an AI algorithm on a large amount of data.

1) *Step 1. Data preparation:* In this step, independent data files to be processed in parallel are identified and put on a list. In the DPAI application, these are the WSIs in the test dataset to be tested on a trained DLNN model.
2) *Step 2. Data preprocessing:* This step includes any data preprocessing, such as data normalization, prior to the primary analysis in the next step. In the DPAI application, the WSI file is too big to be processed directly and has to be split into patch images. These images are organized in the HDF5 file format suitable for parallel file I/O. Note that this preprocessing can be launched as independent tasks in parallel on the HPC cluster.
3) *Step 3. Data processing:* In this primary analysis step, the preprocessed data are processed using the primary analysis application (`APP2`) in parallel tasks across the HPC cluster in a hierarchical and scalable manner—each of the array job tasks (in the first hierarchical layer) launches its own parallel array job tasks (in the second hierarchical layer) and produces results in parallel. In the DPAI application, a color normalization procedure is applied to image patches on the fly and then the trained deep neural network is applied to each color-normalized patch to produce a heatmap.

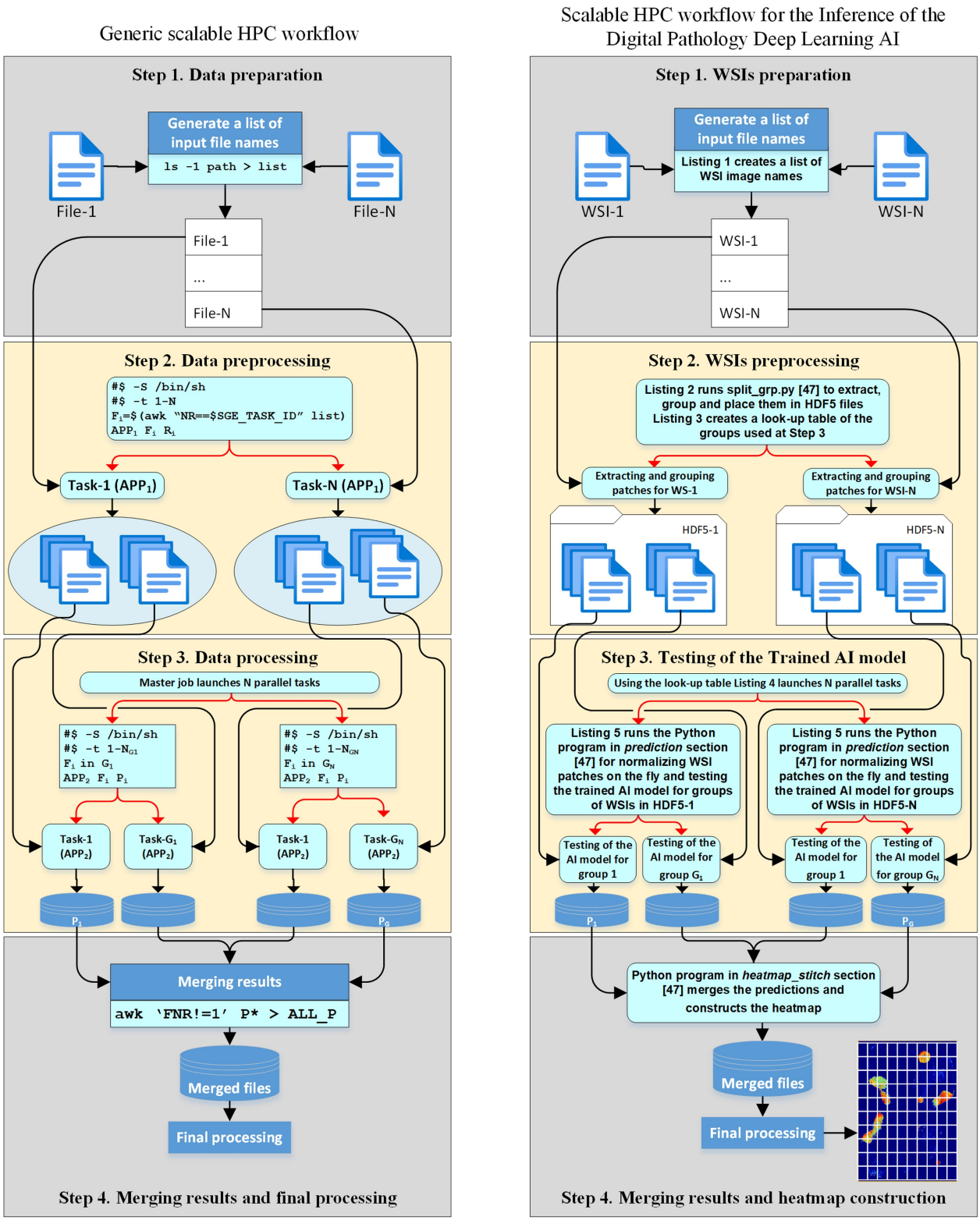

Fig. 8. Generic scalable HPC workflow and its application to the inference of the digital pathology deep learning AI. Highlighted steps contain scalable (parallel) processes, black lines denote data flow, and red lines denote control flow.

4) *Step 4. Merging results and final processing:* This step collects the results produced in parallel and merges them in some application-specific meaningful way. In the DPAI application, the heatmaps for patches are stitched into a WSI-size heatmap that can, for example, be displayed overlapping the original WSI image to indicate the suspicious areas.

VI. Conclusion and Future Work

As the demand for deep learning AI is increasing, sophisticated hardware platforms (also called AI accelerators) are rapidly evolving. While GPU and tensor processing unit remain the mainstream technologies, CPU platforms are considered a suitable and important alternative solution for certain applications [53], especially if a powerful CPU-based HPC facility is already available. MIT professors Dr. Nir Shavit, Dr. Alex Matveev, and others offered and patented a technology to run ML models on CPUs [54], [55], [56]. The practically "unlimited" memory of CPU clusters means these machines could unlock larger problems and architectures than GPUs and other specialty hardware devices. Our techniques could be combined with this technology to provide an end-to-end solution.

The techniques presented in this work do not require any special tools for their implementation. Publicly available and widely used open-source software (Linux, Python, OpenSlide, and HDF5 libraries) along with our source codes written in familiar Linux shell, SGE scripts, and Python programming language and available on GitHub [47] could be used on any HPC cluster with an SGE job scheduler for applying these techniques to similar problems. SGE scripts could also be easily adapted to run under other job schedulers, such as SLURM, PBS, and MOAB.

In conclusion, we developed a technique for scaling pixelwise testing of trained deep learning AI models on HPC clusters. We demonstrated our technique on a digital pathology WSI application. Our technique involved using the HDF5 file format to deal with the parallel I/O of a large number of patch images and the hierarchical scaling and parallelization of the computation jobs using HPC's array job facilities. Our technique is generally applicable to any large-scale image analysis problem involving computationally intensive AI/ML or other analyses on a large number of patch images. We demonstrated our techniques in this article and also shared our source code on Github [47].

Appendix

LISTING 1: A Linux Shell Script Code Excerpt.

```
QSUB=/opt/sge_root/bin/lx-amd64/qsub
ls -1 $FILE_DIR > $LISTFILE
LAST=`cat $LISTFILE | wc -l`
$QSUB -t 1-"$LAST" $ARRAY_SCRIPT
```

This code excerpt creates a list of WSI names and launches an SGE script for processing the WSIs using this list.

LISTING 2: An SGE Script Code Excerpt.

```
#$ -cwd
#$ -S /bin/sh
#$ -j y
FILE=$(awk "NR==$SGE_TASK_ID" $LISTFILE)
time python split_grp.py $FILE
```

This SGE script code excerpt extracts and groups image patches.

LISTING 3: A Linux Shell Script Pseudo-Code.

```
num_files=0
begin=0
end=0

echo "Start End Name" > "$LOOKUP_FILE"

for all output files in the split directory:
do
    num=`awk 'NR==1{print $1}' $file`
    begin=$((num_files+1))
    end=$(($num_files + $num))
    base=`basename "$file"`
    row="$begin $end ${base%.*}"
    echo "$row" >> $LOOKUP_FILE
    num_files=$(($num_files + $num))
done
```

This Linux shell script pseudo-code creates the look-up table.

LISTING 4: Master Job Code Excerpt.

```
#$ -cwd
#$ -S /bin/sh
#$ -j y
#$ -o sysout_main
#$ -N master_job
QSUB=/opt/sge_root/bin/lx-amd64/qsub
ARRAY_SCRIPT=process_array.sh

{read; # skip the first (title) line
   while read LINE; do
     ST=$(echo $LINE | awk -F' ' '{printf $1}')
     EN=$(echo $LINE | awk -F' ' '{printf $2}')
     SUBDIR=$(echo $LINE| awk -F' ' '{printf $3}')
     HDF5_FILE="$SUBDIR"".h5"
     IMG_DIR="$BASE_DIR"/"$SUBDIR"
     SYSOUT_DIR="$IMG_DIR"/sysout
     mkdir -p $SYSOUT_DIR
     HEATMAP_DIR="$IMG_DIR"/preds
     mkdir -p $HEATMAP_DIR
     LOG_DIR="$IMG_DIR"/log_files
     mkdir -p $LOG_DIR

     $QSUB -t "$ST"-"$EN" -o $SYSOUT_DIR -N \
"$SUBDIR"_"$TYPE" $ARRAY_SCRIPT $SPLIT_BASE_DIR \
$HDF5_FILE $HEATMAP_DIR $LOG_DIR
   done
} < $LOOKUP_FILE
```

This master Job code excerpt launches array jobs dynamically.

LISTING 5: A Code Excerpt From process_array.sh.

```
#$ -cwd
#$ -S /bin/sh
#$ -j y
#$ -pe thread 7

GROUP=$(($SGE_TASK_ID - $SGE_TASK_FIRST + 1))

DIR=$1
HDF5_FILE=$2
HEATMAP_DIR=$3
export LOG_DIR=$4

PROG=process_images_grp_normalization_wli.py
time python $PROG $DIR $HDF5_FILE $GROUP \
$HEATMAP_DIR
```

This script is called from the master job dynamically for each HDF5 file to process groups.

ACKNOWLEDGMENT

Disclaimer: The authors would like to thank Dr. Nicholas Petrick and Dr. Kenny Cha of FDA for their valuable feedback that greatly improved the manuscript. This study used the computational resources of the HPC clusters at the FDA, Center for Devices and Radiological Health (CDRH). The authors would also like to thank the anonymous reviewers for their constructive suggestions on highlighting the general applicability of the techniques presented in this article and improving the clarity of the manuscript.

REFERENCES

[1] C. L. Srinidhi, O. Ciga, and A. L. Martel, "Deep neural network models for computational histopathology: A survey," *Med. Image Anal.*, vol. 67, 2021, Art. no. 101813.

[2] B. Chen, T. Medini, J. Farwell, S. Gobriel, C. Tai, and A. Shrivastava, "Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems," in *Proc. Mach. Learn. Syst.*, 2020, vol. 2, pp. 291–306.

[3] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3523–3542, Jul. 2022.

[4] L. Jiao et al., "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, Sep. 5, 2019, doi: 10.1109/ACCESS.2019.2939201.

[5] X. Ren, L. Bo, and D. Fox, "RGB-(D) scene labeling: Features and algorithms," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 2759–2766, doi: 10.1109/CVPR.2012.6247999.

[6] R. Yao, G. Lin, S. Xia, J. Zhao, and Y. Zhou, "Video object segmentation and tracking: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 4, pp. 1–47, 2020.

[7] A. BenTaieb and G. Hamarneh, "Deep learning models for digital pathology," Oct. 27, 2019, *arXiv:1910.12329*.

[8] E. Yildirim and D. J. Foran, "Parallel versus distributed data access for gigapixel-resolution histology images: Challenges and opportunities," *IEEE J. Biomed. Health Inform.*, vol. 21, no. 4, pp. 1049–1057, Jul. 2017.

[9] M. Folk et al., "An overview of the HDF5 technology suite and its applications," in *Proc. EDBT/ICDT March 2011 Workshop Array Databases*, ACM USA, 2011, pp. 36–47, doi: 10.1145/1966895.1966900.

[10] P. Braam, "The Lustre storage architecture," *arXiv:1903.01955*, Mar. 5, 2019. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1903/1903.01955.pdf

[11] F. B. Schmuck and R. L. Haskin, "GPFS: A shared-disk file system for large computing clusters," *FAST*, vol. 2, no. 19, Jan. 28, 2002. [Online]. Available: https://cse.buffalo.edu/faculty/tkosar/cse710_spring14/papers/gpfs.pdf

[12] R. Stone, "py-wsi," 2020, Accessed: Mar. 28, 2020. [Online]. Available: https://github.com/ysbecca/py-wsi

[13] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on GPUs," *J. Syst. Architecture*, vol. 99, 2019, Art. no. 101635, doi: 10.1016/j.sysarc.2019.101635.

[14] J. Keuper and F. Preundt, "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability," in *Proc. 2nd Workshop Mach. Learn. HPC Environments*, 2016, pp. 19–26, doi: 10.1109/MLHPC.2016.006.

[15] S. Pal et al., "Optimizing multi-GPU parallelization strategies for deep learning training," *IEEE Micro,* vol. 39, no. 5, pp. 91–101, Sep./Oct. 2019, doi: 10.1109/MM.2019.2935967.

[16] H. Li, R. Zhao, and X. Wang, "Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification," 2014, *arXiv, abs/1412.4526*.

[17] F. Tschopp, J. N. P. Martel, S. C. Turaga, M. Cook, and J. Funke, "Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems," in *Proc. IEEE 13th Int. Symp. Biomed. Imag.*, 2016, pp. 1225–1228, doi: 10.1109/ISBI.2016.7493487.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3431–3440, 2015. [Online]. Available: https://arxiv.org/pdf/1411.4038.pdf

[19] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Cham, Switzerland: Springer, 2015, pp. 234–241.

[20] F. Xing, Y. Xie, X. Shi, P. Chen, Z. Zhang, and L. Yang, "Towards pixel-to-pixel deep nucleus detection in microscopy images," *BMC Bioinf.*, vol. 20, no. 1, 2019, Art. no. 472, doi: 10.1186/s12859-019-3037-5.

[21] A. Pezeshk, S. Hamidian, N. Petrick, and B. Sahiner, "3-D convolutional neural networks for automatic detection of pulmonary nodules in chest CT," *IEEE J. Biomed. Health Inform.*, vol. 23, no. 5, pp. 2080–2090, Sep. 2019.

[22] D. Tellez, G. Litjens, J. van der Laak, and F. Ciompi, "Neural image compression for gigapixel histopathology image analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 2, pp. 567–578, Feb. 2021.

[23] C. F. Sabottke and B. M. Spieler, "The effect of image resolution on deep learning in radiography," *Radiol., Artif. Intell.*, vol. 2, no. 1, 2020, Art. no. e190015, doi: 10.1148/ryai.2019190015.

[24] P. Balaji, W. Gropp, T. Hoefler, and R. Thakur, "Advanced MPI programming," Tutorial at SC17, Nov. 2017. Accessed: Mar. 2, 2020. [Online]. Available: https://www.mcs.anl.gov/∼thakur/sc17-mpi-tutorial/slides.pdf

[25] S. Amdani and S. Jadhao, "Scheduling jobs strategies for grid computing: A review," *Int. J. Adv. Technol. Eng. Exploration*, vol. 3, no. 19, pp. 82–85, 2016, doi: 10.19101/IJATEE.2016.320002.

[26] A. Reuther et al., "Scheduler technologies in support of high performance data analysis," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Sep. 13, 2016, pp. 1–6.

[27] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Comput.*, vol. 37, no. 9, pp. 633–652, 2011, doi: 10.1016/j.parco.2011.05.005.

[28] E. Deelman et al., "The evolution of the Pegasus workflow management software," *Comput. Sci. Eng.*, vol. 21, no. 4, pp. 22–36, 2019, doi: 10.1109/MCSE.2019.2919690.

[29] N. Zerbe, P. Hufnagl, and K. Schlüns, "Distributed computing in image analysis using open source frameworks and application to image sharpness assessment of histological whole slide images," *Diagn. Pathol.*, vol. 6, no. 1, pp. 1–5, Dec. 2011.

[30] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, "Comparative performance analysis of Intel (R) Xeon Phi (TM) GPU and CPU: A case study from microscopy image analysis," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, 2014, pp. 1063–1072.

[31] X. Qi et al., "Content-based histopathology image retrieval using CometCloud," *BMC Bioinf.*, vol. 15, no. 1, 2014, Art. no. 287.

[32] G. Bueno et al., "A parallel solution for high resolution histological image analysis," *Comput. Methods Programs Biomed.*, vol. 108, no. 1, pp. 388–401, 2012.

[33] A. Aji et al., "Hadoop GIS: A high performance spatial data warehousing system over MapReduce," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1009–1020, 2013.
[34] Lustre Best Practices, Apr. 15, 2019. Accessed: Mar. 21 2020. [Online]. Available: https://www.nas.nasa.gov/hecc/support/kb/lustre-best-practices_226.html
[35] M. Graziani et al., "Breast histopathology with high-performance computing and deep learning," *Comput. Inform.*, vol. 39, no. 4, pp. 780–807, 2020, doi: 10.31577/cai_2020_4_780.
[36] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing deep learning recommender systems' training on CPU cluster architectures," in *Proc. SC20, Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Atlanta, GA, USA, 2020, pp. 1–15.
[37] G. Litjens et al., "1399 H&E-stained sentinel lymph node sections of breast cancer patients: The CAMELYON dataset," *GigaScience*, vol. 7, no. 6, 2018, Art. no. giy065.
[38] B. E. Bejnordi et al., "Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer," *JAMA*, vol. 318, no. 22, pp. 2199–2210, 2017.
[39] Y. Liu et al., "Artificial intelligence-based breast cancer nodal metastasis detection: Insights into the black box for pathologists," *Arch. Pathol. Lab. Med.*, vol. 143, no. 7, pp. 859–868, 2019, doi: 10.5858/arpa.2018-0147-OA.
[40] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/tsmc.1979.4310076.
[41] A. Goode, B. Gilbert, J. Harkes, D. Jukic, and M. Satyanarayanan, "OpenSlide: A vendor-neutral software foundation for digital pathology," *J. Pathol. Inform.*, vol. 4, no. 1, pp. 27–27, Jan. 2013.
[42] A. Vahadane et al., "Structure-preserving color normalization and sparse stain separation for histological images," *IEEE Trans. Med. Imag.*, vol. 35, no. 8, pp. 1962–1971, Aug. 2016, doi: 10.1109/TMI.2016.2529665.
[43] W. Li and W. Chen, "Reproducibility in deep learning algorithms for digital pathology applications: A case study using the CAMELYON16 datasets," in *Proc. SPIE*, vol. 11603, 2021, Art. no. 1160318, doi: 10.1117/12.2581996.
[44] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep learning for identifying metastatic breast cancer," Jun. 18, 2016, *arXiv:1606.05718*. [Online]. Available: https://arxiv.org/pdf/1606.05718.pdf
[45] A. Collette, *HDF5 for Python*, 2008. [Online]. Available: http://www.h5py.org/
[46] *Son of Grid Engine*, 2018. Accessed: Mar. 27, 2020. [Online]. Available: https://arc.liv.ac.uk/trac/SGE
[47] "HPC_DPI," 2020. Accessed: Jun. 30, 2020. [Online]. Available: https://github.com/DIDSR/HPC_DPAI
[48] "Grand Challenge," 2015. Accessed: Mar. 26, 2020. [Online]. Available: https://camelyon16.grand-challenge.org/
[49] M. Mikailov et al., "Scaling bioinformatics applications on HPC," *BMC Bioinf.*, *vol.* 18, pp. 163–169, 2017, doi: 10.1186/s12859-017-1902-7.
[50] H. Luo et al., "DRAR-CPI: A server for identifying drug repositioning potential and adverse drug reactions via the chemical–protein interactome," *Nucleic Acids Res.*, vol. 39, pp. W492–W498, Jul. 2011, doi: 10.1093/nar/gkr299.
[51] M. Mikailov, J. Qiu, F.-J. Luo, S. Whitney, and N. Petrick, "Scaling modeling and simulation on high-performance computing clusters," *Simulation*, vol. 96, no. 2, pp. 221–232, 2020, doi: 10.1177/0037549719878249.
[52] M. Mikailov et al., "Scaling and parallelization of big data analysis on HPC and cloud systems," in *Proc. Int. Conf. Adv. Comput. Commun. Eng.*, 2019, pp. 1–8, doi: 10.1109/ICACCE46606.2019.9079987.
[53] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," Jul. 24, 2019, *arXiv:1907.10701*. [Online]. Available: https://arxiv.org/pdf/1907.10701.pdf
[54] A. Matveev and N. Shavit, "Systems and methods for exchange of data in distributed training of machine learning algorithms." US Patent App. 16/193,051, 2019/5/23. Accessed: Apr. 17, 2020. [Online]. Available: https://patents.google.com/patent/US20190156214A1/en
[55] R. Gelashvili, N. Shavit, and A. Zlateski, "L3 fusion: Fast transformed convolutions on CPUs," *arXiv:1912.02165*, Dec. 4, 2019. [Online]. Available: https://arxiv.org/pdf/1912.02165.pdf
[56] "Software-Delivered AI - Neural Magic," 2019. Accessed: Apr. 17, 2020. [Online]. Available: https://neuralmagic.com/

Weizhe Li received the B.S. degree in bioengineering from Sichuan University, Chengdu, China, in 2002, the Ph.D. degree in biophysics from the Chinese Academy of Sciences, Beijing, China, in 2008, and the MPH degree in biostatistics from the University of Maryland, College Park, MD, USA, in 2018.

From 2018 to 2014, he was a Postdoctoral Fellow with Johns Hopkins University. From 2014 to 2019, he was an Optical Imaging Research Specialist with the National Institutes of Health. He is currently a Staff Fellow with the Center for Devices and Radiological Health, U.S. Food & Drug Administration, Silver Spring, MD, USA. He has authored/coauthored 11 multidisciplinary articles in peer-reviewed journals and holds a patent for a three-dimensional imaging technique. His research interests include machine learning in digital pathology and whole slide imaging.

Dr. Li was the recipient of awards and honors including the Leidos Research Fellowship, the Mid-Atlantic Nephrology Young Investigator's Forum Award, and Diao Scholarship (Chinese Academy of Sciences).

Mike Mikailov (Member, IEEE) received the B.S. and M.Sc. degrees in mathematics from Baku State University, (Baku, Azerbaijan), former USSR, in 1980, the Ph.D. degree in computer science from the Glushkov Institute of Cybernetics, Academy of Sciences of Ukraine, Kyiv, Ukraine, in 1987, and the MBA degree in international management from the Naveen Jindal School of Management, University of Texas at Dallas, Richardson, TX, USA, in 2006.

From October 2004 to March 2009, he was the Director of Software Development with L-3 Communications. Since May 2009, he has been a Computer Scientist with the Center for Devices and Radiological Health, U.S. Food & Drug Administration, Silver Spring, MD, USA. He has authored/coauthored more than 20 articles in peer-reviewed journals and conference proceedings in former USSR and 9 articles in peer-reviewed journals in the USA. His current research interests include artificial intelligence and machine learning algorithms in digital pathology and other medical imaging applications, high-performance computing techniques, and parallel and distributed computing techniques.

Dr. Mikailov was the recipient of many awards and honors, including the Bronze Medal for Contributions to the Scientific and Technical Progress in Computer Science (World Festival of Youth and Students, Moscow, former USSR, 1985), the Scientific Achievement Award (US FDA, 2011), the Leveraging/Collaboration Award "For development of a new reference viral database to enhance next-generation sequencing analysis for novel virus detection" (US FDA, 2018), and Honor Award (US FDA, 2020).

Weijie Chen received the B.S. degree in physics from the University of Science and Technology of China, Hefei, China, in 1998, the M.Sc. degree in medical physics from Peking University, Beijing, China, in 2001, and the Ph.D. degree in medical physics from the University of Chicago, Chicago, IL, USA, in 2007.

Since 2007, he has been a Postdoctoral Fellow, a Staff Fellow, and a Research Physicist with the Center for Devices and Radiological Health (CDRH), U.S. Food & Drug Administration, Silver Spring, MD, USA. He has authored/coauthored more than 65 articles in peer-reviewed journals and conference proceedings and 3 book chapters and holds 1 patent. He has been the Principal Investigator or Coinvestigator in numerous FDA intramural funded research projects. His current research interests include artificial intelligence and machine learning algorithms in digital pathology and other medical imaging applications and statistical evaluation methodologies in diagnostic medicine.

Dr. Chen is a Fellow of SPIE and a Member of the American Association of Physicists in Medicine. He is an Associate Editor for the *SPIE Journal of Medical Imaging*. He was the recipient of many awards and honors including the Predoctoral Traineeship Award (US Department of Defense Breast Cancer Research Program, 2004–2006), the Lawrence H. Lanzl Medical Physics Graduate Student Award (University of Chicago, 2005), Excellence in Scientific Research Award (CDRH, FDA, 2017), and the Excellence in Mentoring Award (CDRH, FDA, 2020).