Original Article

# LibMI: An Open Source Library for Efficient Histopathological Image Processing

**Yuxin Dong[1], Pargorn Puttapirat[1], Jingyi Deng[1], Xiangrong Zhang[2], Chen Li[1]**

[1]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, Shaanxi, China, [2]Institute of Intelligent Information Processing, Xidian University, Xi'an, Shaanxi, China

Submitted: 15-Feb-2020 Revised: 20-Apr-2020 Accepted: 25-Jun-2020 Published: 21-Aug-2020

## Abstract

**Background:** Whole-slide images (WSIs) as a kind of image data are rapidly growing in the digital pathology domain. With unusual high resolution, these images make them hard to be supported by conventional tools or file formats. Thus, it obstructs data sharing and automated analysis. Here, we propose a library, LibMI, along with its open and standardized image file format. They can be used together to efficiently read, write, modify, and annotate large images. **Materials and Methods:** LibMI utilizes the concept of pyramid image structure and lazy propagation from a segment tree algorithm to support reading and modifying and to guarantee that both operations have linear time complexity. Further, a cache mechanism was introduced to speed up the program. **Results:** LibMI is an open and efficient library for histopathological image processing. To demonstrate its functions, we applied it to several tasks including image thresholding, microscopic color correction, and storing pixel-wise information on WSIs. The result shows that libMI is particularly suitable for modifying large images. Furthermore, compared with congeneric libraries and file formats, libMI and modifiable multiscale image (MMSI) run 18.237 times faster on read-only tasks. **Conclusions:** The combination of libMI library and MMSI file format enables developers to efficiently read and modify WSIs, thus can assist in pixel-wise image processing on extremely large images to promote building image processing pipeline. The library together with the data schema is freely available on GitLab: https://gitlab.com/BioAI/libMI.

**Keywords:** Extremely large image, image processing, open format, whole-slide image

## Introduction

Digitizing pathology requires biomedical informatic tools which could facilitate storage and delivery of visual data which pathologists usually observe via a light microscope. The current challenge is that conventional image file systems cannot support the wide range of functionalities required for reading and writing of extremely large images. Over the last decade, the presence of whole-slide images (WSIs) in digital pathology drives the development of various open and proprietary file formats and tools. In 2013, introduction of OpenSlide[1] marks the establishment of mainstream file formats, which has been widely adopted to store histopathological images, and becomes a *de facto* standard in digital pathology. Nevertheless, to our knowledge, none of the files readable by OpenSlide and other file formats offer ways to modify existing files and create a new one. A typical uncompressed WSI with full resolution can range between 1 and 20 GB in file size,[2,3] and a typical compressed WSI may easily take 1 GB of storage.[4] The monstrous file size makes them relatively slow to process using conventional techniques. This tremendously increases computational burden and causes difficulties in providing holistic features for intelligent software. Thus, currently there are still limitations to modify WSIs. It remains to be a technical bottleneck obstructing data scientists to fully exploit the information and potentials in these images.[5]

The existing solutions separate a WSI into a number of smaller tiles using read-only libraries, e.g., OpenSlide[1] or Bio-Formats,[6] before processing by conventional image processing tools and libraries, e.g., OpenCV and Sci-kit

Access this article online

Quick Response Code:

Website: www.jpathinformatics.org

DOI: 10.4103/jpi.jpi_11_20

**Address for correspondence:** Prof. Chen Li, School of Electronic and Information Engineering, Xi'an Jiaotong University, Xianning West Road, Xi'an Shaanxi, 710049 China. E-mail: cli@xjtu.edu.cn



**How to cite this article:** Dong Y, Puttapirat P, Deng J, Zhang X, Li C. LibMI: An open source library for efficient histopathological image processing. J Pathol Inform 2020;11:26.

Available FREE in open access from: http://www.jpathinformatics.org/text.asp?2020/11/1/26/292718

image.[7-11] Due to extremely large file size and image dimension, it is difficult to efficiently access WSIs while maintaining a decent reading and especially modifying speed. Furthermore, these images have different proprietary formats created by different WSI scanner vendors,[1] e.g., Aperio (.svs, .tif) and Hamamatsu (.vms, .vmu, .ndpi). Each vendor provides a proprietary software to view the WSIs generated by its own scanners. The existence of proprietary file formats and software for viewing and analyzing presents three problems. First, the file formats directly obstruct free data sharing and curation of open data in this domain. Second, the proprietary viewing and analysis software prevents the development of general analysis pipelines which can be modified for different purposes. They also cannot be extended by existing open source image processing libraries. Third, the *ad hoc* solutions cannot support easy reuse of existing annotations and the interoperation between analysis software.[12,13] Large image processing lacks an universal and comprehensive solution.

In this paper, we present libMI, an open source multiscale image library for manipulating WSIs. It is compatible with all proprietary WSI file formats and can read, write, and modify extremely large multiscale images at any resolutions. It also supports both pixel-wise geometrical and semantic annotations, e.g., regional boundaries and cancer grading in histopathological images. Along with the library, we present an open format called modifiable multiscale image (MMSI) to store large images based on SQLite with the library to access this format. Seamlessly working with libMI, the novel format supports efficient regional modification of extremely large images without the necessity of updating the entire image or cutting the image to tiles. The library works as a robust and efficient abstract layer for the proposed data format, and this is the first efficient implementation of reading and writing WSIs, thus can be used to enhance performance of relevant libraries or standards.[14-16] Note that the library is not limited to WSIs but also capable to deal with large images in other domains, such as satellite and high-resolution panoramic images. The library and file format focus on a frequently encountered problem in image analysis, especially those artificial intelligent (AI) systems of computer vision in all domains.

## Materials and Methods

### The libMI library

LibMI was designed to read, write, modify, and annotate the WSI files. It treats each WSI file as a libMI project, which includes original image, labeling matrix, labeling table, and meta-data, as shown in Figure 1. The mentioned objects would be saved as a folder with relevant files in it to provide portability and interoperability. Currently, we are using OpenSlide to read the image data from various proprietary digital slides, and further, modification can be made to support other large image formats such as DICOM, while hiding the working details of complex low-level systems, such as the organization of the data structure and the algorithms to process these data from the user.

The library stores all important information in a single JSON file, which is a lightweight, text-based, language-independent data-interchange format for the portable representation of structured data.[17] This file contains all parameters needed to process the image, as well as image meta-data which are the properties of the original WSI file. Most vendors provide WSI files that contain various properties, such as the number of down-sampled layers, available z-stack layers, and scanning resolution. LibMI provides public Application Programming Interface (APIs) to access existing properties and add new properties, which can be useful when new meta-data must be saved.

The labeling matrix is stored in the MMSI open format that we proposed in this paper and is managed by our underlying library, which is based on the standardized SQLite schema. The matrix has the same dimension as the main image, with each pixel in the matrix describing which region each pixel belongs to with a unique ID number. Pixels with the same value can be recognized as being in the same region even though they may not be connected.

Labeling table contains the definition of each matrix region which is linked by ID number. Since there could be billions of different regions, the labeling table is also divided into smaller sections, compressed by the DEFLATE algorithm, and saved as blobs in SQLite. The blobs are dynamically created when needed to save disk space. The possible value in the labeling table is up to 255 for every region ID.

Intuitively, the proposed mechanism that incorporates MMSI labeling matrix with labeling table may seem to be redundant. Nevertheless, it is proposed to provide both efficiency and flexibility since MMSI matrix is strictly structured for read and write speed, and labeling table allows changes in the number of categories and addition in the description of each region. The combination of the labeling matrix and the labeling table provides the capability to annotate regions of any shape on the image and to give each region a corresponding label. Since the files in libMI projects are all standardized, including only SQLite and JSON, it is compatible with libraries or tools to directly access the project's data without the libMI library.

### The modifiable multiscale image file format

Since existing file formats cannot accommodate dynamically changing and complex data as needed in the WSIs, we propose MMSI which is an open file format to store extremely large images. It works with libMI library to store labeling matrices to support pixel-wise annotation. It can also store the WSI files to give high-efficiency read, write, and modify access of these images. MMSI stores images as a tile-based pyramidal structure, containing several layers with different sizes. The lowest layer has the same dimension as the original image, and the following adjacent layer has half of both the width and height of the previous layer. Every $2 \times 2$ tiles in one layer can be directly mapped to a single tile in the upper layer as depicted in Figure 2a. Each layer is divided into tiles with the method as shown in Figure 2b. Each single tile is stored as a

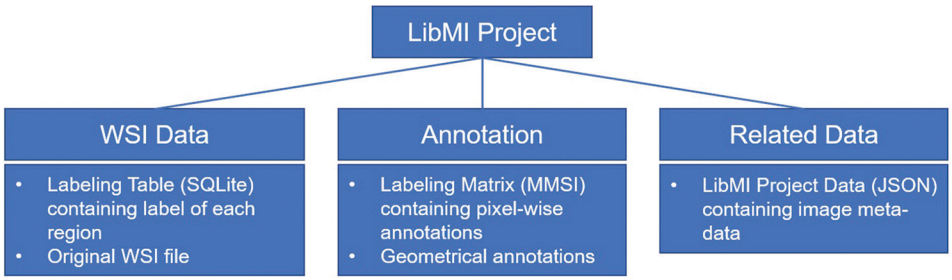


**Figure 1:** The libMI project organization. Each project contains three components: (1) Whole-slide image data which include the original whole-slide image file and the labeling table stored in an SQLite file, (2) annotations which include geometrical annotations and pixel-wise semantic annotation labeling matrix stored in an modifiable multi-scale image file, and (3) related data which include the image meta-data stored in a JSON file

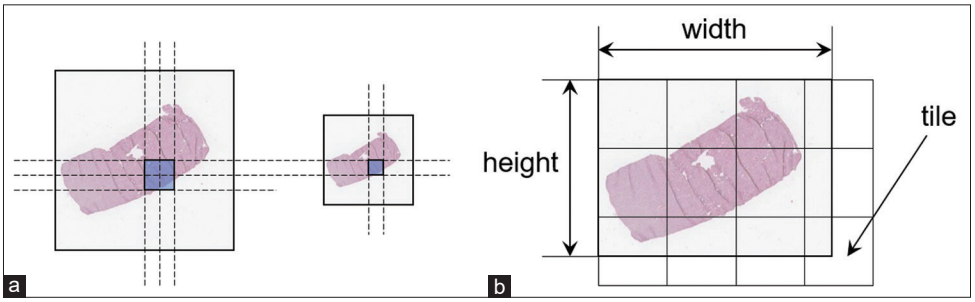


**Figure 2:** Tiling mechanism of modifiable multiscale image. (a) 2 × 2 tiles in a higher layer can be directly mapped to one tile in the lower layer. (b) The schematic diagram of tiling method of a layer in the pyramid

blob in SQLite (https://www.sqlite.org/). MMSI can use PNG or JPG format as the internal compression method, which is selected by the user.

Currently, the ability of MMSI format to partially modify large images is only used to make pixel-wise annotation, which is the main feature provided by libMI. Actually, this ability can benefit any process that requires modifications of data in large images, including general image preprocessing, noise reduction, image enhancement, and color value correction.

## Underlying algorithms of modifiable multiscale image

### *Forward updating and backward updating*

To efficiently read and modify the images, we utilize the concept of two-dimensional segment tree and lazy propagation to minimize the calculation complexity. Each tile in the pyramid can be treated as a node on the tree, and each node has up to four children. Reading and modifying the tiles are the same as making queries and modifications. In this way, we can guarantee the upper limit of the number of tiles visited in each operation.

Forward updating and backward updating are the two essential operations in the processing algorithms. The former one means passing the modification from one node to its children and resetting the lazy value, while another one means passing the modification from the node's children to itself. The time complexity of both operations is O (1).

## Reading and modifying the image

Since WSIs are very large, it is impossible to load the whole image into the memory, so only parts of the image would be accessed by the library. In different situations, one may need to get a thumbnail of the whole image or get detailed information in a small region. The resolution of region of interest (ROI) varies significantly in these two cases, but the actual resolutions the users need are limited by viewing hardware – computer displays. The concept of pyramid image is exploited to minimize the amount of data needed to read from hard disks.

For each reading or modifying operation, two input parameters are required: A ROI to operate, and the actual resolution needed. Then, the appropriate downsample ratio (DSR) will be automatically calculated. Each layer has its own DSR, and the algorithm selects the preferred layer from them according to the following criterion: the layer with the highest DSR but still lower than the one requested by the user. In this way, we could get the output image with virtually no quality loss and guarantee the complexity to be minimized.

Reading and modifying arbitrary regions of the labeling matrix are the two main operations provided by the MMSI processing library. After selecting the preferred layer, the system starts from the top layer which has the lowest resolution. Forward updating is applied to every tile in the top layer. Iterating through each above the preferred layer, both operations would have the linear complexity of O (4 n/3). Figure 3 shows the schematic diagram.

In both operations, forward updating needs to be applied to the lazy valued tiles before accessing matrix data in the ROI. All lazy valued tiles which are in the ROI and above the preferred

layer as the colored regions as shown in Figure 3a will be updated. Then, according to the type of requested operation, the algorithm will either read image data from the preferred layer or overwrite image data in the preferred layer.

Modifying operation requires extra steps. The mechanism of lazy propagation can only guarantee that the modifications applied to higher layers are passed to lower layers. To update modifications in the opposite direction, backward updating is applied to all tiles in the green regions layer by layer from the bottom to the top after red regions as shown in Figure 3b have been modified in each modifying operation.

### The caching mechanism

The bottleneck between the memory and the SQL database is caused by intensive compression and read and write operations. To alleviate the bottleneck, libMI records the times that each tile has been visited and uses a priority queue to determine which tile should be removed from cache, according to the Least Frequently Used Strategy. Size of the cache can be changed at runtime via libMI API to adjust the balance between memory occupation and processing speed.

### Parallel processing

LibMI supports parallel processing in each operation. Unlike OpenSlide and Bio-Formats that require developers to use multithreads explicitly by themselves, libMI hides all implementation details from users and allows them to access the file in a serial manner, as well as gain benefit from a parallel processing mechanism provided by libMI.

## Results

### Overview

The proposed library, libMI, is capable of manipulating extremely large images and compatible with all OpenSlide-compatible WSI files,[1] regardless of hardware limitations. It supports instantaneous reading, writing, and modifying image data of any region in any resolution without being forced to cut image into tiles or update an entire image and also recording pixel-wise geometrical and semantic annotations, such as cancer subtypes or gradings. It is written in C++ and also officially provides public standardized programmatic APIs for Python. We analyzed the upper limit of time complexity in each operation, which will take no more than 800 ms in normal scenarios when the processing resolution does not exceed 10 megapixels, which is more than resolution needed to fill 4K computer displays at 8.3 megapixels. LibMI could be used on different platforms including Linux, macOS, and Windows. Along with the libMI library, MMSI is the open format we proposed which is capable of storing any kind of large images with any resolution. LibMI and MMSI are both free and open source. The proposed open data format is built on other open formats including SQLite for the storage of large multiscale images and JSON for accompanying information about the WSI; thus, it can be accessed via not only libMI but also other tools as well. More information about the guideline and the openness is at libMI documentation: https://bioai.gitlab.io/libMI-docs/.

### Performance

The library is tested on an Intel Core i7-9750H CPU (2.60 GHz) with 16 GB RAM and 1 TB SSD under Windows 10 Operating System. We select four WSIs from The Cancer Genome Atlas (TCGA) with different dimensions to show the performance of MMSI processing different file sizes.[18] Table 1 shows the files used and their sizes, including the original WSI file, the file after converting to MMSI with JPG and PNG internal compression method, respectively, and the total size of all image tiles exported from the WSI saved in PNG format. It shows that MMSI using JPG compression method is slightly larger than the original file, while MMSI using PNG compression method is significantly larger. Therefore, MMSI using JPG is more suitable to store WSIs for read-only access after preprocessing, and MMSI using PNG is more suitable to store pixel-wise annotation data as these data are easier to compress. The time spent to convert a WSI into MMSI is listed in Table 2. When converting, image data are read and uncompressed by OpenSlide and then compressed again and written into MMSI tile by tile.

Figure 4 shows the relationship between the required resolution and processing time of both the reading and modifying operations accessing an MMSI file using PNG compression with a typical WSI dimension (80,000, 80,000). It shows the performance when running with cache size of 4000 tiles, or

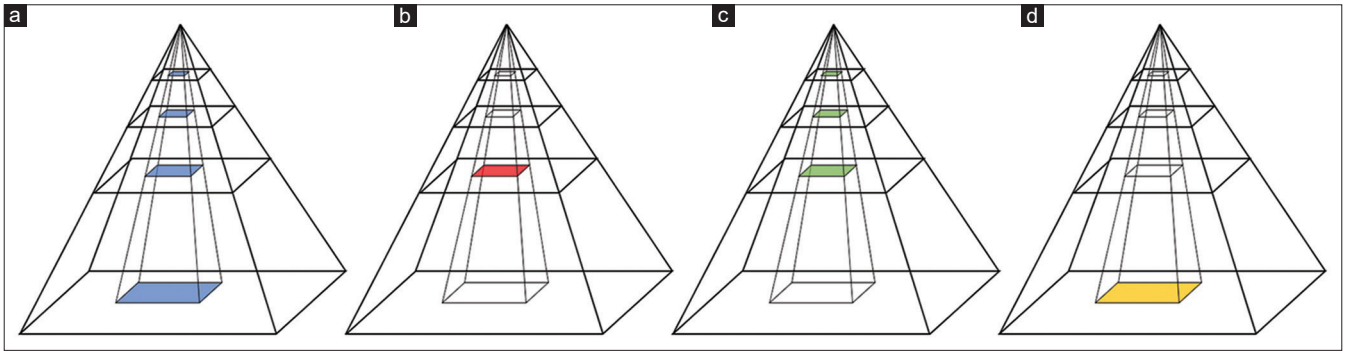


**Figure 3:** Schematic diagram of a reading or modifying operation. (a) Region of interest in the pyramid structure. (b) Region of interest in the preferred layer. (c) Tiles in region of interest needed to be accessed in this operation. (d) Tiles in region of interest but irrelevant to this operation

**Table 1: Whole-slide images files used in performance testing and their sizes in megabytes**

| ID | File name | Original | MMSI (JPG) | MMSI (PNG) | Raw PNG |
|---|---|---|---|---|---|
| 1 | TCGA-BP-5201-01Z-00-DX1 | 337 | 557 | 7649 | 7640 |
| 2 | TCGA-BP-4771-01Z-00-DX1 | 806 | 966 | 12,173 | 12,136 |
| 3 | TCGA-B0-5098-01Z-00-DX1 | 1034 | 1346 | 16,089 | 15,998 |
| 4 | TCGA-BP-4176-01Z-00-DX1 | 1174 | 1496 | 18,450 | 18,168 |

MMSI: Modifiable multiscale image

**Table 2: Time spent to convert whole-slide images into modifiable multiscale images in seconds (s)**

| ID | File name | MMSI (JPG) | MMSI (PNG) |
|---|---|---|---|
| 1 | TCGA-BP-5201-01Z-00-DX1 | 534 | 664 |
| 2 | TCGA-BP-4771-01Z-00-DX1 | 753 | 1036 |
| 3 | TCGA-B0-5098-01Z-00-DX1 | 1258 | 1802 |
| 4 | TCGA-BP-4176-01Z-00-DX1 | 1082 | 1477 |

MMSI: Modifiable multiscale image

1 GB of RAM. In normal circumstances where the number of pixels processed does not exceed 10 megapixels, the processing time is at most around 800 ms. Note that in some circumstances, the processing time of reading operations can exceed that of modifying ones. It is because of the utilization of lazy propagation so that modifications applied do not take effect immediately, while the results are still guaranteed to be correct. In these experiments, reading or modifying regions are randomly selected, and in real-world scenarios, the regions are likely to be continuous,[19] so the cache mechanism will be utilized to gain even higher performance.

Besides the capability of efficiently modifying WSIs, which is not supported by any other tools, libMI can also achieve higher performance when providing read-only access to WSIs compared to congeneric software. The library is tested to perform read-only tasks on the four WSI files from TCGA, and the processing time is compared with other two WSI reading libraries: OpenSlide and Bio-Formats.[1,6] We randomly generated 100 reading requests for each WSI file and tested the speed using libMI to read MMSI files converted from proprietary files, as well as using OpenSlide and Bio-Formats to read these original files. We tested the performance of MMSI with different compression methods, different cache sizes, and different number of threads.

The average reading speed for each WSI file is recorded in Tables 3 and 4. The result shows that MMSI using PNG is 18.237 times faster than OpenSlide and 32.473 times faster than Bio-Formats in average, and MMSI using JPG is 40.621 times faster than OpenSlide and 70.921 times faster than Bio-Formats in average, while doing the same reading job. LibMI and MMSI together have gained significant speed advantage over congeneric libraries and file formats.

## Use Case I: Applying conventional image operations on whole-slide images

In medical image processing, thresholding algorithms could generate a binary image according to a source image and a given threshold. It is one of the essential operations used to analyze the image data because it can effectively separate the foreground from the background. Further, color correction or normalization is another algorithm that is often applied to microscopic images to standardize the color representation. This example uses the Otsu algorithm for thresholding and histogram matching method for color correction.[20,21] Sample WSIs come from TCGA,[18] as shown in Figure 5.

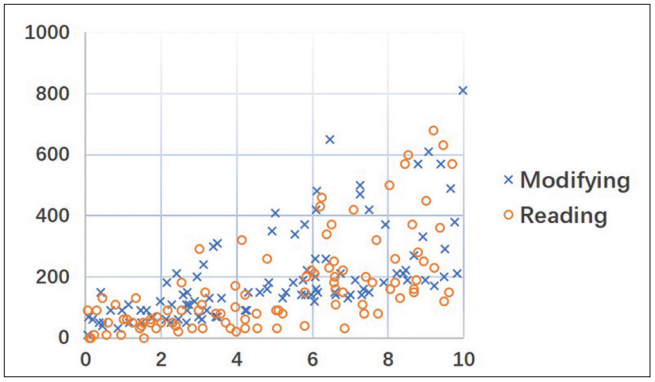


**Figure 4:** Processing time of libMI with cache size 16,000. X-axis is the requested resolution (megapixels) and Y-axis is the processing time (ms)

For thresholding, the example program iterates through tiles of the image using libMI library to obtain the intensity distribution of the image and to calculate the threshold and then iterates again to apply thresholding to the source image. The resulting image is stored in the MMSI format, and a thumbnail from a downsampled layer is obtained through libMI library.

For color correction, the gist is similar. The program first iterates both the source and target images to obtain the histogram and then iterates the target image again to apply color correction, store the result in MMSI format, and get the final thumbnail.

For normal images, these operations would not raise technical issues. However, for WSI, the operation must be applied to a small part of an image at a time due to hardware limitations. With libMI, developers can easily access image data of the WSIs tile by tile through the API of libMI library without any preprocessing and conveniently write the result image as another WSI file for further operations.

## Usage Case II: Freehand pixel-wise annotation

Pixel-wise reading and writing annotations are one of the major features proposed by libMI. This allows annotators to add freehand annotations and image processing algorithms to perform image segmentation. Figure 6 shows the result of

**Table 3: Average processing speed in megapixels and equivalent megabytes with different cache sizes**

| WSI ID | 1 | | 2 | | 3 | | 4 | |
|---|---|---|---|---|---|---|---|---|
| Cache | PNG | JPG | PNG | JPG | PNG | JPG | PNG | JPG |
| 0 | | | | | | | | |
| MP/s | 273.67 | 288.26 | 222.50 | 410.84 | 242.82 | 512.50 | 226.30 | 496.30 |
| MB/s | 1094.70 | 1153.04 | 889.99 | 1643.37 | 971.28 | 2050.00 | 905.20 | 1985.20 |
| 1GB | | | | | | | | |
| MP/s | 389.45 | 925.91 | 569.69 | 1419.40 | 740.15 | 1291.23 | 534.89 | 1382.45 |
| MB/s | 1557.79 | 3703.64 | 2278.75 | 5677.60 | 2960.61 | 5164.92 | 2139.57 | 5529.81 |
| 2GB | | | | | | | | |
| MP/s | 534.82 | 979.34 | 645.73 | 1589.59 | 903.00 | 1497.12 | 893.28 | 1458.06 |
| MB/s | 2139.29 | 3917.34 | 2582.92 | 6358.37 | 3612.00 | 5988.47 | 3573.13 | 5832.23 |
| 4GB | | | | | | | | |
| MP/s | 630.23 | 1003.07 | 813.11 | 1729.14 | 1074.34 | 1629.58 | 1226.28 | 1608.21 |
| MB/s | 2520.91 | 4012.26 | 3252.42 | 6916.58 | 4297.37 | 6518.32 | 4905.10 | 6432.83 |

MP/s: Megapixels, MB/s: Megabytes, WSI: Whole-slide images

**Table 4: Average processing speed in megapixels and equivalent megabytes with different thread numbers**

| Number of threads | | 1 | | 2 | | 4 | | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| Method | WSI ID | MP/s | MB/s | MP/s | MB/s | MP/s | MB/s | MP/s | MB/s |
| MMSI (PNG) | 1 | 120.83 | 483.33 | 214.24 | 856.95 | 395.04 | 1580.18 | 630.23 | 2520.91 |
| | 2 | 212.63 | 850.53 | 346.30 | 1385.22 | 671.24 | 2684.94 | 813.11 | 3252.42 |
| | 3 | 250.97 | 1003.89 | 485.30 | 1941.20 | 726.84 | 2907.35 | 1074.34 | 4297.37 |
| | 4 | 238.58 | 954.30 | 365.88 | 1463.51 | 660.27 | 2641.08 | 1226.28 | 4905.10 |
| MMSI (JPG) | 1 | 231.51 | 926.03 | 368.04 | 1472.18 | 872.20 | 3488.78 | 1003.07 | 4012.26 |
| | 2 | 337.04 | 1348.18 | 688.40 | 2753.60 | 1233.98 | 4935.93 | 1729.14 | 6916.58 |
| | 3 | 407.96 | 1631.84 | 782.38 | 3129.52 | 1307.79 | 5231.14 | 1629.58 | 6518.32 |
| | 4 | 375.96 | 1503.84 | 659.43 | 2637.72 | 1393.99 | 5575.98 | 1608.21 | 6432.83 |
| OpenSlide | 1 | 10.81 | 43.24 | 20.82 | 83.28 | 37.55 | 150.18 | 50.51 | 202.04 |
| | 2 | 9.70 | 38.82 | 17.69 | 70.75 | 30.22 | 120.88 | 42.68 | 170.73 |
| | 3 | 11.97 | 47.88 | 21.73 | 86.94 | 33.92 | 135.67 | 46.54 | 186.16 |
| | 4 | 12.22 | 48.88 | 17.14 | 68.57 | 30.27 | 121.06 | 40.76 | 163.06 |
| Bio-Formats | 1 | 5.49 | 21.98 | 11.49 | 45.94 | 19.61 | 78.45 | 24.32 | 97.28 |
| | 2 | 4.15 | 16.60 | 8.39 | 33.57 | 15.74 | 62.95 | 21.37 | 85.48 |
| | 3 | 4.22 | 16.88 | 7.59 | 30.38 | 14.82 | 59.26 | 20.83 | 83.34 |
| | 4 | 4.99 | 19.94 | 9.25 | 36.99 | 16.88 | 67.54 | 22.09 | 88.34 |

MP/s: Megapixels, MB/s: Megabytes, WSI: Whole-slide images, MMSI: Modifiable multi-scale image

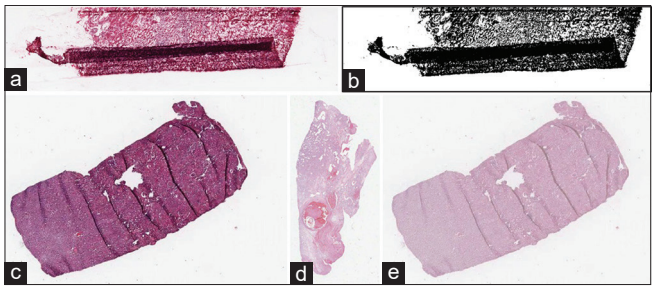


**Figure 5:** Whole-slide image processed by conventional algorithms with assistances from libMI. (a and b) Source and result image in thresholding, (c-e) source image, target image, and result image of color correction, respectively

two consecutive writing operations which create overlapping freehand regions. The fact that libMI API allows developers to access the labeling matrix at any region and resolution makes pixel-wise modifications of labeling matrix possible. Intuitively, the modifications are made at the level where the annotator is currently viewing, and the modifications will be passed to other levels with different resolutions automatically. In the overlapping area of two annotations, the newer annotations would overwrite the former ones, for example (a) would be overwritten by the latter one (b) in Figure 6. This example has demonstrated that the library is able to automatically choose the proper layer to satisfy the accessing resolution and pass the modifications between different layers.

## Code availability

Both the libMI library and MMSI open format are freely available at https://gitlab.com/BioAI/libMI under GNU General Public License v3.0, and the documentation for both libraries is available at https://bioai. gitlab.io/libMI-docs/.

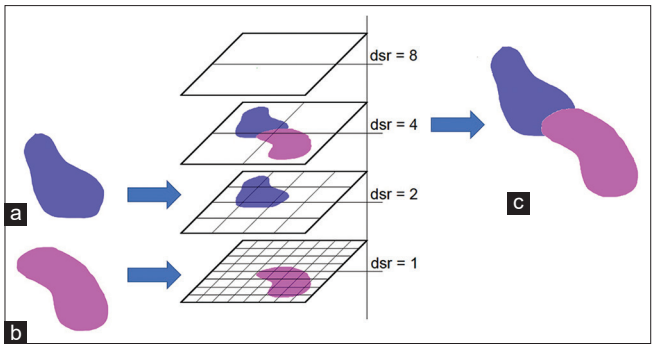


**Figure 6:** The procedure of doing freehand pixel-wise annotation in libMI. The test program writes two freehand annotations (a and b) to two different lower layers in the modifiable multiscale image pyramidal structure and read the result (c) from a higher layer (DSR is downsample ratio)

## Discussion

Since there are no strict definitions of extremely large image, except its dimension, the description of types of image array which MMSI file format can support should be clarified. In theory, the dimension of extremely large images can be indefinitely large; however, different aspects of hardware limitations which establish a working environment which software should follow include disk read/write speed, memory size, graphic-processing unit processing throughput, and computer displays. In usual, we consider an image file larger than 6 GB when uncompressed, which is (40,000, 40,000) dimension with 32-bit pixels, as an extremely large image. As an anchor, we suggest consideration of displayable pixels which will generally reflect the display resolution at a given displaying cycle (image frame). The current reference we adopt is 4 K resolution, approved by The International Telecommunication Union (ITU), which has the processing resolution around 8.3 megapixels. Another aspect to consider is image bit-depth which has industry standard at 8-bit, 16-bit, and 24-bit per channel. To preserve compatibility with existing viewing tools and data format when parts of extremely large image are requested to be viewed, both libMI and MMSI could store image with higher bit depth at 64-bit for one pixel, so the color model can be either grayscale, true-color with alpha channels, or other widely used color models, e.g., ARGB, HSV, and CIE L*a*b*. The production of extremely large images can be made through two approaches. First, taking several normal-sized images and stitching them together such as those from WSI scanners or aerial imaging. Second, the image produced from a very high-resolution image sensor. In either case, images saved should not only be small but also highly accessible so that they can be utilized efficiently.

The current version of libMI has achieved decent performance. We believe that the performance could be improved if the following limitations are addressed. The first is the reading and writing bottleneck. The libMI performance relies on fast disk reading and writing speed, and most of the processing time was taken by these operations. This problem is currently being addressed by a caching mechanism in libMI. The second is the compression time. In the cases that many sequential compression tasks are needed, the performance of the library may be affected. This problem is addressed by parallel processing mechanism in libMI. Finally, the performance of the library can benefit from better hardware with superior disk response time and read/write speed. LibMI uses the PNG lossless compression by default to avoid image quality deterioration after multiple modifying operations. Nevertheless, the JPEG lossy compression can also be utilized if only one-off modification is required to minimize the WSI file size.[22]

Previous works in pathology have shown the potentials and possibilities of WSIs in automatic screening, diagnosis, and treatment planning of cancer patients.[7,23] The centralized large-scale biomedical repositories hosting WSIs such as The Cancer Genome Atlas Project (TCGA) and Genotype-Tissue Expression Project (GTEx) have emerged,[18,24] and the performance of the repository distributing images to clients could be improved by letting viewers only access parts of WSIs without transferring entire files. Furthermore, efforts to tie nonimaging information in digital images have been made in numerous competitions, e.g., CAMELYON, TUPAC, and ICIAR.[25-27] In those competitions, evaluations and developments of new and existing algorithms for automatic detection of cancer metastases in hematoxylin and eosin-stained WSIs were proposed. Not losing the medical meta-data along the way could benefit further analysis in the future.

LibMI and MMSI may be implemented in all computational environments: Cloud servers and local machines. Currently, there are a number of WSI-compatible tools available for utilization including OpenHI, ASAP, QuPath, Cytomine, OpenSeadragon, and SlideJ.[12,13,28-31] These frameworks are successful implementations of computational pathology to support visualization, annotation, and further pathological analysis. Unfortunately, many of them were being forced to only support proprietary WSI file formats since the free and open ones are not available. Extending the read/write engine of the mentioned framework to include libMI would allow the image, annotations, and associated clinical data to be saved in a single unified file. Pathological e-learning resource database such as the Stanford Tissue Microarray Database and the "digital lung pathology" could be upgraded by adopting libMI as well.[32,33] This would allow them to overcome the current shortcomings such as fixed magnification and limited number of views. In implementation, there are no limits to how the proposed library and file format could be utilized.

LibMI also helps to save time and disk space in machine learning tasks which include WSIs. The conventional preprocessing method of WSI is to extract patches – fixed size small image tiles – which has the same dimensions as the input of analysis models; then, the annotation will be generated according to the tiles, e.g., image segmentation. This results in redundant disk space consumption. With the decent reading speed, libMI enables these machine learning algorithms to read

image data from the WSIs directly during training, as well as pixel-wise annotation data. The functionality of pixel-wise annotation can also avoid the time consumed on converting soft overlay annotations. The only information that needs to be stored is the position of each patch, which is rather small compared to image data.

## Conclusions

Currently, no tools are available for some essential operations in WSIs, especially to partially modify WSI in a subregion at different accessing levels. The combination of libMI library and MMSI file format resolves the problems since they provide simplified access to the complex file organization such as multiscale image and accompanying data. Furthermore, since the libMI project organization and MMSI file format are standardized and open, it can be accessed by anyone with other tools. To our knowledge, no open format is currently in use and can support efficient storage and modification of extremely large images. Therefore, libMI encourages the sharing of intermediate and final analysis results. In addition, libMI can significantly reduce the time complexity of large image modifications because of partial file modification and efficient updating algorithms, unlike other open image file formats such as BigTIFF, which requires the entire file to be rewritten. The performance and efficiency of libMI are further enhanced by characteristics of SQLite, compression algorithm, cache mechanism, and parallel processing, resulting in higher reading efficiency compared to congeneric software or libraries. With the mentioned advantages, libMI enables easier sharing, modification, and writing of large image data.

Since libMI is compatible with existing proprietary WSI formats, it can be integrated to existing systems without compatibility problems. Building analysis pipelines on libMI should be more straightforward since there is no need for complex preprocessing which transforms large images into smaller patches and keeping them in separated files. LibMI can promote the development of automated pipelines and the application of artificial intelligence in various domains, especially in pathology, since analyzing histopathological images is the key to assist automated diagnosis.

### Financial support and sponsorship

This work has been supported by The National Natural Science Foundation of China (61772409); The National Key Research and Development Program of China (2018YFC0910404); The consulting research project of the Chinese Academy of Engineering (The Online and Offline Mixed Educational Service System for "The Belt and Road" Training in MOOC China); Project of China Knowledge Centre for Engineering Science and Technology; The innovation team from the Ministry of Education (IRT_17R86); and the Innovative Research Group of the National Natural Science Foundation of China (61721002).

### Conflicts of interest

There are no conflicts of interest.

## References

1. Goode A, Gilbert B, Harkes J, Jukic D, Satyanarayanan M. OpenSlide: A vendor-neutral software foundation for digital pathology. J Pathol Inform 2013;4:27.
2. Boyce BF. An update on the validation of whole slide imaging systems following FDA approval of a system for a routine pathology diagnostic service in the United States. Biotech Histochem 2017;92:381-9.
3. Evans AJ, Bauer TW, Bui MM, Cornish TC, Duncan H, Glassy EF, *et al*. US Food and Drug Administration approval of whole slide imaging for primary diagnosis: A key milestone is reached and new questions are raised. Arch Pathol Lab Med 2018;142:1383-7.
4. Singh R, Chubb L, Pantanowitz L, Parwani A. Standardization in digital pathology: Supplement 145 of the DICOM standards. J Pathol Inform 2011;2:23.
5. Tabata K, Mori I, Sasaki T, Itoh T, Shiraishi T, Yoshimi N, *et al*. Whole-slide imaging at primary pathological diagnosis: Validation of whole-slide imaging-based primary pathological diagnosis at twelve Japanese academic institutes. Pathol Int 2017;67:547-54.
6. Linkert M, Rueden CT, Allan C, Burel JM, Moore W, Patterson A, *et al*. Metadata matters: Access to image data in the real world. J Cell Biol 2010;189:777-82.
7. Coudray N, Ocampo PS, Sakellaropoulos T, Narula N, Snuderl M, Fenyö D, *et al*. Classification and mutation prediction from non-small cell lung cancer histopathology images using deep learning. Nat Med 2018;24:1559-67.
8. Cruz-Roa A, Gilmore H, Basavanhally A, Feldman M, Ganesan S, Shih NN, *et al*. Accurate and reproducible invasive breast cancer detection in whole-slide images: A deep learning approach for quantifying tumor extent. Sci Rep 2017;7:46450.
9. Yu KH, Zhang C, Berry GJ, Altman RB, Ré C, Rubin DL, *et al*. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. Nat Commun 2016;7:12474.
10. Bradski, G. The OpenCV library. Dobbs J Software Tools 2000;4:2236121.
11. van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, *et al*. scikit-image: Image processing in Python. PeerJ 2014;2:e453.
12. Bankhead P, Loughrey MB, Fernández JA, Dombrowski Y, McArt DG, Dunne PD, *et al*. QuPath: Open source software for digital pathology image analysis. Sci Rep 2017;7:16878.
13. Litjens G. Automated Slide Analysis Platform (ASAP); 2015. Available from: https://github.com/computationalpathologygroup/ASAP. [Last accessed on 2019 Jul 20].
14. Marée R, Rollus L, Stévens B, Hoyoux R, Louppe G, Vandaele R, *et al*. Collaborative analysis of multi-gigapixel imaging data using Cytomine. Bioinformatics 2016;32:1395-401.
15. Herrmann MD, Clunie DA, Fedorov A, Doyle SW, Pieper S, Klepeis V, *et al*. Implementing the DICOM standard for digital pathology. J Pathol Inform 2018;9:37.
16. Marques Godinho T, Lebre R, Silva LB, Costa C. An efficient architecture to support digital pathology in standard medical imaging repositories. J Biomed Inform 2017;71:190-7.
17. Crockford D. JSON: The fat-free alternative to XML; 2006. Available from: http://www.json.org/fatfree.html. [Last accessed on 2019 Jul 20].
18. Tomczak K, Czerwińska P, Wiznerowicz M. The Cancer Genome Atlas (TCGA): An immeasurable source of knowledge. Contemp Oncol (Pozn) 2015;19:A68-77.
19. Walkowski S, Lundin M, Szymas J, Lundin J. Students' performance during practical examination on whole slide images using view path tracking. Diagn Pathol 2014;9:208.
20. Cheriet M, Said JN, Suen CY. A recursive thresholding technique for image segmentation. IEEE Trans Image Process 1998;7:918-21.
21. Nyúl LG, Udupa JK, Zhang X. New variants of a method of MRI scale standardization. IEEE Trans Med Imaging 2000;19:143-50.
22. Zarella MD, Bowman D, Aeffner F, Farahani N, Xthona A, Absar SF, *et al*. A practical guide to whole slide imaging: A white paper from the digital pathology association. Arch Pathol Lab Med 2019;143:222-34.
23. Signaevsky M, Prastawa M, Farrell K, Tabish N, Baldwin E, Han N,

*et al*. Artificial intelligence in neuropathology: Deep learning-based assessment of tauopathy. Lab Invest 2019;99:1019-29.
24. Lonsdale J, Thomas J, Salvatore M, Phillips R, Lo E, Shad S, *et al*. The genotype-tissue expression (GTEx) project. Nat Genet 2013;45:580-5.
25. Ehteshami Bejnordi B, Veta M, Johannes van Diest P, van Ginneken B, Karssemeijer N, Litjens G, *et al*. Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer. JAMA 2017;318:2199-210.
26. Veta M, Heng YJ, Stathonikos N, Bejnordi BE, Beca F, Wollmann T, *et al*. Predicting breast tumor proliferation from whole-slide images: The TUPAC16 challenge. Med Image Anal 2019;54:111-21.
27. Aresta G, Araújo T, Kwok S, Chennamsetty SS, Safwan M, Alex V, *et al*. Bach: Grand challenge on breast cancer histology images. Med Image Anal 2019;56:122-39.
28. Puttapirat P, Zhang H, Lian Y, Wang C, Zhang X, Yao L, *et al*. OpenHI-An open source framework for annotating histopathological image. BIBM 2018 IEEE International Conference on Bioinformatics and Biomedicine; 2018. p. 1076-82.
29. Marée R, Rollus L, Stévens B, Hoyoux R, Louppe G, Vandaele R, *et al*. Cytomine: An open-source software for collaborative analysis of whole-slide images. Diagn Pathol 2016;1:13.
30. OpenSeadragon Project; 2013. Available from: https://github.com/openseadragon/openseadragon. [Last accessed on 2020 May 07].
31. Della Mea V, Baroni GL, Pilutti D, Di Loreto C. SlideJ: An ImageJ plugin for automated processing of whole slide images. PLoS One 2017;12:e0180540.
32. Marinelli RJ, Montgomery K, Liu CL, Shah NH, Prapong W, Nitzberg M, *et al*. The Stanford tissue microarray database. Nucleic Acids Res 2008;36:D871-7.
33. Kayser K, Kayser G, Radziszowski D, Oehmann A. From telepathology to virtual pathology institution: The new world of digital pathology. Rom J Morphol Embryol 1999;45:3-9.