
KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2023

KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT

China 2023

Develop, Build & Deploy Lightweight AI Services with WebAssembly

- *Kinfey Lo*
- *Jianbai Ye*
- *Vivian Hu*

Who are us

kinfey Lo

Microsoft Senior Cloud Advocate , Focus on Data + AI , Traveling around spreading technology

Jianbai Ye

Semedia Tech CEO, help people to build and spread their own knowledge-based agent.

Vivian Hu

Product Manager at Second State, building the next-gen could infra with Rust and WebAssembly. Organize Rust and WebAssembly meetup in Asia.

Agenda

- Why Wasm and AI
- WASI NN and WasmEdge introduction
- AI inference demo: Mediapipe
- AI inference demo: Llama2
- Demo: serverless AI agents

Scan this QR code to access all the resources

Why Wasm and AI

- **Lightweight.** The Wasm inference application itself is often a few MBs instead of GBs for Python.
- **High performance**. The Wasm app itself performs as fast as native binaries, making it feasible to implement pre- and post-processing tasks for inference in the application itself without complex dependencies.
- **Portability.** The Wasm app is write once run anywhere. It automatically takes advantage of all the native hardware features available on the device it is deployed on.
- **Safe and cloud-ready.** The Wasm app runs in a secure sandbox that can be seamlessly managed by container tools, such as Docker and Kubernetes.
- **Polyglot.** Wasm applications can be written in multiple languages including C/C++, Rust, Go, JavaScript or even Python itself!
- **Easy to develop.** The Wasm inference application has simple and straightforward dependencies. It is very easy to develop and has a very simple (and hence secure) software supply chain.

WASI-NN and WasmEdge Introduction

What is WASI-NN?

WASI-NN stands for Neural Network proposal for WASI, aiming to introduce a group of simple yet powerful APIs to bridge the Deep Learning inference and WebAssembly with existing models.

WASI-NN and WasmEdge Introduction

What is WASI-NN?

```
let encoding = wasi_nn::GRAPH_ENCODING_...;
let target = wasi_nn::EXECUTION_TARGET_CPU;
let graph = wasi_nn::load(&[bytes, more_bytes], encoding, target);

// Configure the execution context.
let context = wasi_nn::init_execution_context(graph);
let tensor = wasi_nn::Tensor { ... };
wasi_nn::set_input(context, 0, tensor);

// Compute the inference.
wasi_nn::compute(context);
wasi_nn::get_output(context, 0, &mut output_buffer, output_buffer.len());
```

WASI-NN and WasmEdge Introduction

WasmEdge Runtime

A CNCF Sandbox project

WasmEdgeRuntime

https://github.com/WasmEdge/WasmEdge

CLOUD NATIVE
COMPUTING FOUNDATION

WASI-NN and WasmEdge Introduction

Fork 609 | Starred 6.7k

About

WasmEdge is a lightweight, high-performance, and extensible WebAssembly runtime for cloud native, edge, and decentralized applications. It powers serverless apps, embedded functions, microservices, smart contracts, and IoT devices.

WasmEdgeRuntime

Contributors 148

+ 137 contributors

WASI-NN and WasmEdge Introduction

Seamlessly managed by existing container tools

WasmEdge features: High performance networking

- Rust
  - tokio + mio
  - hyper and hyper-tls
  - warp
  - reqwest
  - http_req and rustls
  - WasmEdge sockets API
- JavaScript
  - Node.js API for servers and SSR
  - fetch()

WASI-compatible non-blocking I/O connections implemented through epoll

WASI-NN and WasmEdge Introduction

WasmEdge features: Integration with infra services

WASI-NN and WasmEdge Introduction

WasmEdge features: Supported AI frameworks

PyTorch

OpenVINO

TensorFlow

ggml

WASI-NN and WasmEdge Introduction

WasmEdge features: data processing framework

OpenCV

FFMPEG

AI inference: Mediapipe

```rust
use mediapipe_rs::postprocess::utils::draw_detection;
use mediapipe_rs::tasks::vision::ObjectDetectorBuilder;

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let (model_path, img_path, output_path) = parse_args()?;

    let mut input_img = image::open(img_path)?;
    let detection_result = ObjectDetectorBuilder::new()
        .max_results(2) // set max result
        .build_from_file(model_path)? // create a object detector
        .detect(&input_img)?; // do inference and generate results

    // show formatted result message
    println!("{}", detection_result);

    if let Some(output_path) = output_path {
        // draw detection result to image
        draw_detection(&mut input_img, &detection_result);
        // save output image
        input_img.save(output_path)?;
    }

    Ok(())
}
```

AI inference: Mediapipe

- Install WasmEdge and WasmEdge TensorFlow plugin
- Git clone the mediapipe-rs demo project from GitHub:

  git clone https://github.com/juntao/demo-object-detection

- Build the rust project into Wasm file:

  cargo build --target wasm32-wasi --release

- AoT the wasm file to achieve high performance with wasmedge compile:

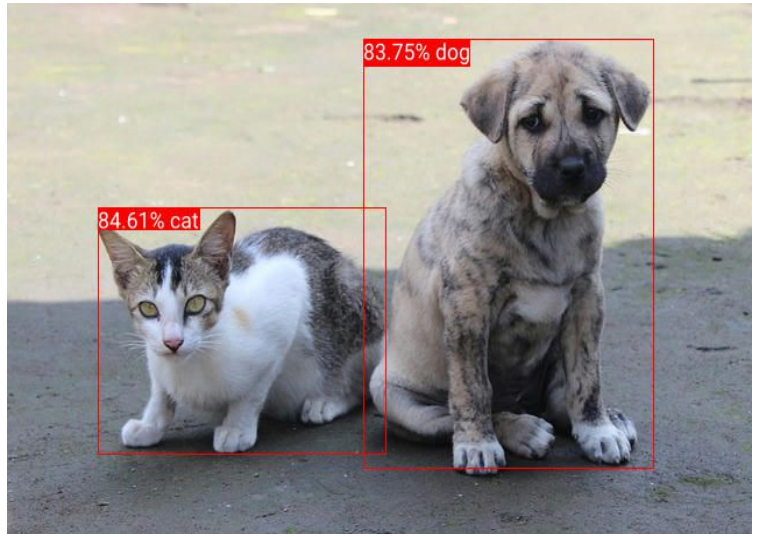
  wasmedge compile ./target/wasm32-wasi/release/demo-object-detection.wasm ./demo-object-detection_aot.wasm

- Run the wasm file:

  wasmedge --dir .:. demo-object-detection-aot.wasm example.jpg output.jpg

AI inference: Mediapipe

```
     Running `/mediapipe-rs/./scripts/wasmedge-runner.sh target/wasm32-wasi/release/
DetectionResult:
  Detection #0:
    Box: (left: 0.12283102, top: 0.38476586, right: 0.51069236, bottom: 0.851197)
    Category #0:
      Category name: "cat"
      Display name:  None
      Score:         0.8460574
      Index:         16
  Detection #1:
    Box: (left: 0.47926134, top: 0.06873521, right: 0.8711677, bottom: 0.87927735)
    Category #0:
      Category name: "dog"
      Display name:  None
      Score:         0.8375256
      Index:         17
```



AI inference: Llama 2


```rust
    let graph =
        wasi_nn::GraphBuilder::new(wasi_nn::GraphEncoding::Ggml, wasi_nn::ExecutionTarget::AUTO)
            .build_from_cache(model_name)
            .unwrap();
    let mut context = graph.init_execution_context().unwrap();

    let system_prompt = String::from("<<SYS>>You are a helpful, respectful and honest assistant. Always answer as short as possible, while being safe. <</SYS>>");
    let mut saved_prompt = String::new();

    loop {
        println!("Question:");
        let input = read_input();
        if saved_prompt == "" {
            saved_prompt = format!("[INST] {} {} [/INST]", system_prompt, input.trim());
        } else {
            saved_prompt = format!("{} [INST] {} [/INST]", saved_prompt, input.trim());
        }

        // Set prompt to the input tensor.
        let tensor_data = saved_prompt.as_bytes().to_vec();
        context
            .set_input(0, wasi_nn::TensorType::U8, &[1], &tensor_data)
            .unwrap();

        // Execute the inference.
        context.compute().unwrap();

        // Retrieve the output.
        let mut output_buffer = vec![0u8; 1000];
        let output_size = context.get_output(0, &mut output_buffer).unwrap();
        let output = String::from_utf8_lossy(&output_buffer[..output_size]).to_string();
        println!("Answer:\n{}", output.trim());

        saved_prompt = format!("{} {} ", saved_prompt, output.trim());
    }
}
```

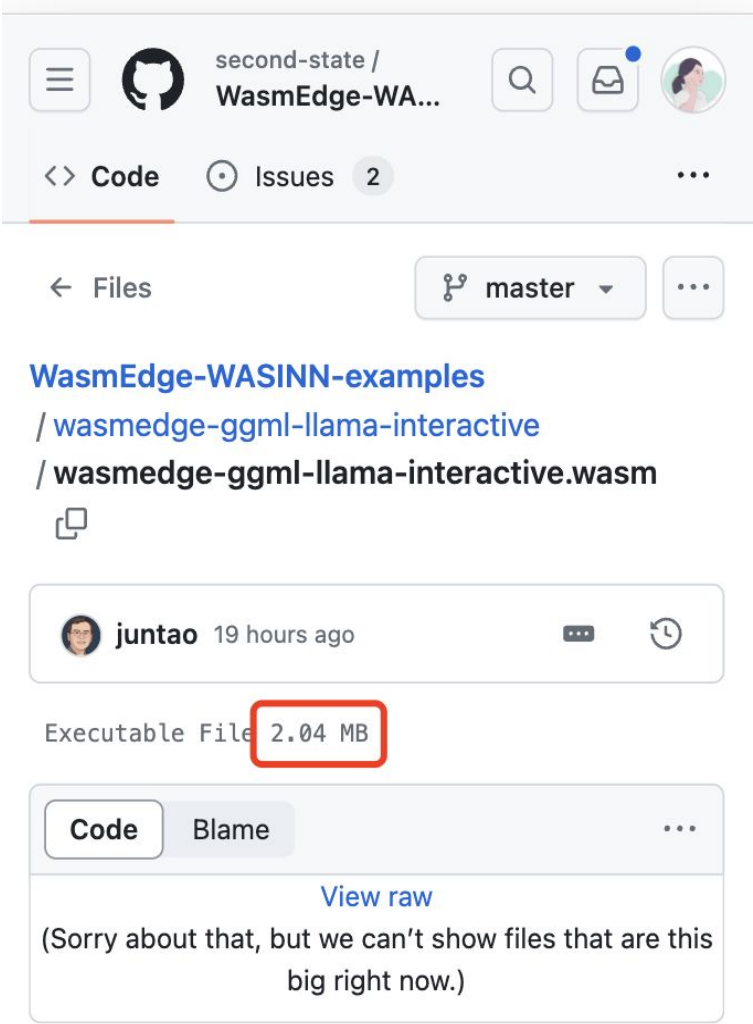
AI inference: Llama 2

```
cargo build --target wasm32-wasi --release

   Compiling proc-macro2 v1.0.66
   Compiling unicode-ident v1.0.11
   Compiling thiserror v1.0.48
   Compiling quote v1.0.33
   Compiling syn v2.0.31
   Compiling thiserror-impl v1.0.48
   Compiling wasi-nn v0.6.0 (https://github.com/second-state/wasmedge-wasi
   Compiling wasmedge-ggml-llama v0.1.0 (/home/ubuntu/llama-wasmedge/WasmE
    Finished release [optimized] target(s) in 3.51s

ubuntu@ip-172-31-0-156:~/llama-wasmedge/WasmEdge-WASINN-examples/wasmedge-ggml-lla
ls -al *.wasm

-rw-rw-r-- 1 ubuntu ubuntu 2236604 Sep  6 04:44 wasmedge-ggml-llama.wasm
```



The application binary is about 2MB.

AI inference: Llama 2

Execute

Execute the WASM with the `wasmedge` using the named model feature to preload large model:

```
wasmedge --dir .:. \
  --nn-preload default:GGML:CPU:llama-2-7b-chat.Q5_K_M.gguf \
  wasmedge-ggml-llama-interactive.wasm default
```

After executing the command, you may need to wait a moment for the input prompt to appear. You can enter your question once you see the `Question:` prompt:

```
Question:
What's the capital of the United States?
Answer:
The capital of the United States is Washington, D.C. (District of Columbia).
Question:
What about France?
Answer:
The capital of France is Paris.
Question:
I have two apples, each costing 5 dollars. What is the total cost of these apples?
Answer:
The total cost of the two apples is $10.
Question:
What if I have 3 apples?
Answer:
The total cost of 3 apples would be 15 dollars. Each apple costs 5 dollars, so 3 apples would cost 3 x 5 = 15 dollar
```

https://github.com/second-state/WasmEdge-WASINN-examples/blob/master/wasmedge-ggml-llama-interactive/

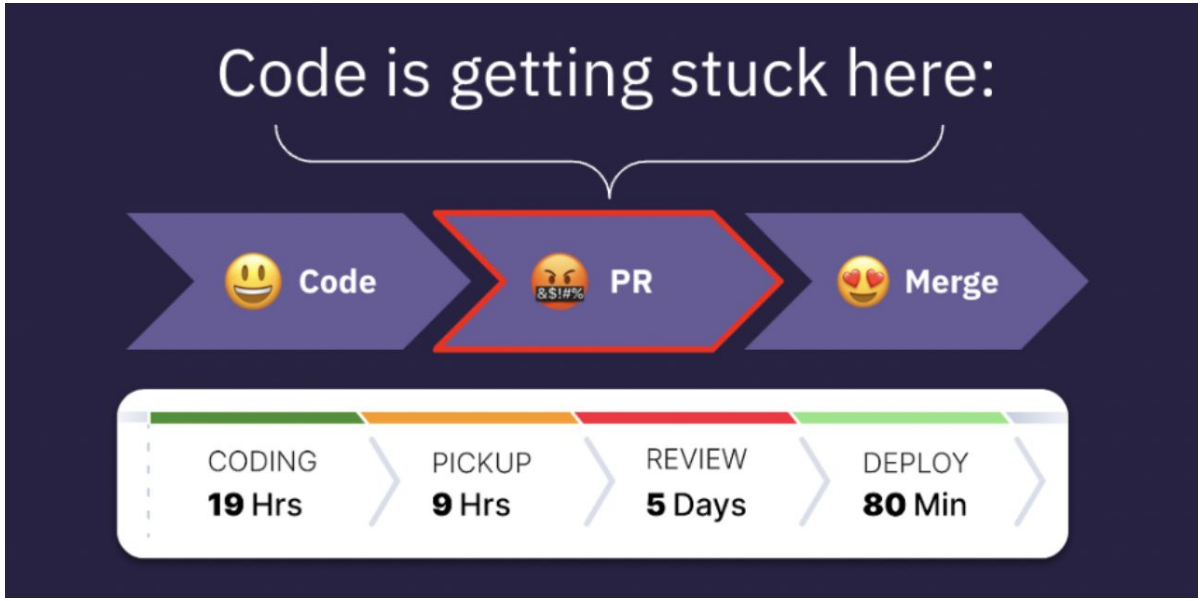
AI inference: Llama 2

- Install WasmEdge and WasmEdge ggml plugin
- Git clone the template project from GitHub: git clone ***https://github.com/second-state/WasmEdge-WASINN-examples***
- Build the rust project into Wasm file: ***cargo build --target wasm32-wasi --release***
- AoT the wasm file to achieve high performance with wasmedge compile
- Run the wasm file

Demo: serverless AI agents

- Senior developers are very busy and very expensive.
- Yet, the development process cannot move forward (eg merging the PR) without the code review.

  Developers are often idle waiting for reviews.



Demo: serverless AI agents

Add function 'check_prime' for node's crypto API by Aviii06 · Pull Request #82

Changes from **all commits** | File filter | Conversations

Filter changed files

- example_js/node
  - main.mjs
  - package.json
  - rollup.config.js
- src

49 src/internal_module/crypto.rs

```
+     if n <= 1 {
+         return JsValue::Bool(false);
+     }
+     let limit = (n as f64).sqrt() as i32;
+     for a in 2..limit {
+         if n % a == 0 {
+             return JsValue::Bool(false);
+         }
+     }
+     JsValue::Bool(true)
+ }
+
```

Demo: serverless AI agents

second-state/wasmedge-quickjs **Add function 'check_prime' for node's crypto API**

Potential problems:

1. The check_prime function can be optimized further, as it checks for divisibility with even numbers after 2, which isn't necessary.

Demo: serverless AI agents


```rust
#[no_mangle]
#[tokio::main(flavor = "current_thread")]
pub async fn run() -> anyhow::Result<()> {
    dotenv().ok();
    logger::init();
    log::debug!("Running github-pr-summary/main");

    let owner = env::var("github_owner").unwrap_or("juntao".to_string());
    let repo = env::var("github_repo").unwrap_or("test".to_string());
    let trigger_phrase = env::var("trigger_phrase").unwrap_or("flows summarize".to_string());

    let events = vec!["pull_request", "issue_comment"];
    listen_to_event(&GithubLogin::Default, &owner, &repo, events, |payload| {
        handler(
            &owner,
            &repo,
            &trigger_phrase,
            payload,
        )
    })
    .await;

    Ok(())
}
```

Step 1: The application registers with a Rust host app to receive external trigger events.

When the event is received, the host will call run() again and listen_to_event() will be able to retrieve the event data in the payload.

Demo: serverless AI agents


```rust
let pulls = octo.pulls(owner, repo);
let patch_as_text = pulls.get_patch(pull_number).await.unwrap();
let mut current_commit = String::new();
let mut commits: Vec<String> = Vec::new();
for line in patch_as_text.lines() {
    if line.starts_with("From ") {
        // Detected a new commit
        if !current_commit.is_empty() {
            // Store the previous commit
            commits.push(current_commit.clone());
        }
        // Start a new commit
        current_commit.clear();
    }
    // Append the line to the current commit if the current commit i
    if current_commit.len() < CHAR_SOFT_LIMIT {
        current_commit.push_str(line);
        current_commit.push('\n');
    }
}
if !current_commit.is_empty() {
    // Store the last commit
    commits.push(current_commit.clone());
}
```

Step 2: The handler() function use GitHub Rust SDK to retrieve all patches associated with commits in the PR.

The PR information is passed to the function via the payload.

Demo: serverless AI agents

```
let chat_id = format!("PR#{pull_number}");
let system = &format!("You are an experienced software developer. You will act as a reviewer for
let mut openai = OpenAIFlows::new();
openai.set_retry_times(3);

let mut reviews: Vec<String> = Vec::new();
let mut reviews_text = String::new();
for (_i, commit) in commits.iter().enumerate() {
    let commit_hash = &commit[5..45];
    log::debug!("Sending patch to OpenAI: {}", commit_hash);
    let co = ChatOptions {
        model: MODEL,
        restart: true,
        system_prompt: Some(system),
    };
    let question = "The following is a GitHub patch. Please summarize the key changes and identi
    match openai.chat_completion(&chat_id, &question, &co).await {
        Ok(r) => {
            if reviews_text.len() < CHAR_SOFT_LIMIT {
                reviews_text.push_str("------\n");
                reviews_text.push_str(&r.choice);
                reviews_text.push_str("\n");
            }
            let mut review = String::new();
            review.push_str(&format!("### [Commit {commit_hash}](https://github.com/WasmEdge/Was
            review.push_str(&r.choice);
            review.push_str("\n\n");
            reviews.push(review);
            log::debug!("Received OpenAI resp for patch: {}", commit_hash);
        }
        Err(e) => {
            log::error!("OpenAI returned an error for commit {commit_hash}: {}", e);
        }
    }
}
```

Step 3: Each patch is sent to ChatGPT for summarization. The commit patch summaries are stored in an array.

Demo: serverless AI agents

```
let mut resp = String::new();
resp.push_str("Hello, I am a [code review bot](https://github.com/flows-network/github-pr-
if reviews.len() > 1 {
    log::debug!("Sending all reviews to OpenAI for summarization");
    let co = ChatOptions {
        model: MODEL,
        restart: true,
        system_prompt: Some(system),
    };
    let question = "Here is a set of summaries for software source code patches. Each summ
    match openai.chat_completion(&chat_id, &question, &co).await {
        Ok(r) => {
            resp.push_str(&r.choice);
            resp.push_str("\n\n## Details\n\n");
            log::debug!("Received the overall summary");
        }
        Err(e) => {
            log::error!("OpenAI returned an error for the overall summary: {}", e);
        }
    }
}
for (_i, review) in reviews.iter().enumerate() {
    resp.push_str(review);
}

// Send the entire response to GitHub PR
// issues.create_comment(pull_number, resp).await.unwrap();
match issues.update_comment(comment_id, resp).await {
    Err(error) => {
        log::error!("Error posting resp: {}", error);
    }
    _ => {}
}
```

Step 4: Use ChatGPT API to summarize the summaries and send the result back to the PR as a comment.

Demo: serverless AI agents

1. Load the template
2. Configure the GitHub connection
3. Configure the OpenAI connection
4. Test if works

Resources

- The full tutorials: https://github.com/second-state/kubecon-eu-2023
- Join WasmEdge Discord server: https://discord.gg/U4B5sFTkFc
- CNCF #WasmEdge Slack Channel: https://slack.cncf.io/#wasmedge
- WasmEdge Twitter: https://twitter.com/realwasmedge
- WasmEdge GitHub repo: https://github.com/WasmEdge/WasmEdge
- flows.network: https://flows.network/

夏歌
中国大陆

扫一扫上面的二维码图案，加我为朋友。