Rust 专场

How does WebAssembly become a preferred runtime for Rust?

Michael Yuan 2023年05月28日

```
$ cargo build --release
    Compiling hello v0.1.0 (/.../rust-examples/hello)
    Finished release [optimized] target(s) in 0.53s

$ ls -al target/release/hello
-...x 2 root root 4322184 ... target/release/hello

$ target/release/hello
Hello WasmEdge!
```

```rust
fn main() {
  let s : &str = "Hello WasmEdge!";
  println!("{}", s);
}
```

```
$ rustup target add wasm32-wasi

$ cargo build --target wasm32-wasi --release
    Compiling hello v0.1.0 (/.../rust-examples/hello)
    Finished release [optimized] target(s) in 0.61s

$ ls -al target/wasm32-wasi/release/hello.wasm
-...x 2 root root 2132806 ... target/.../hello.wasm

$ wasmedge target/wasm32-wasi/release/hello.wasm
Hello WasmEdge!
```


WASM
IS AWESOME
docker
+WASM

Why WebAssembly (Wasm)?

Cross-platform portable

- Write once run anywhere – like Java
- No need to cross-compile
- Only need to port the Wasm runtime to many Oses and hardware platforms
  - Linux / Windows / Mac OS / seL4
  - aarch64, x86, RISC-V

Why WebAssembly (Wasm)?

GOTC

Secure

- Sandboxed memory access
- Capability-based security model to access OS resources
- Safer than traditional Linux containers
  - Proven to support untrusted code
  - Much reduced attack surface

Why WebAssembly (Wasm)?

Manageable by container tools

Why WebAssembly (Wasm)?

GOTC

Embeddable

- Can be embedded into a host application written in a different language
  - Rust
  - Go
  - C/C++
  - Java
  - Python
  - .NET / C#
- Great for creating plugins or extensions

Why WebAssembly (Wasm)?

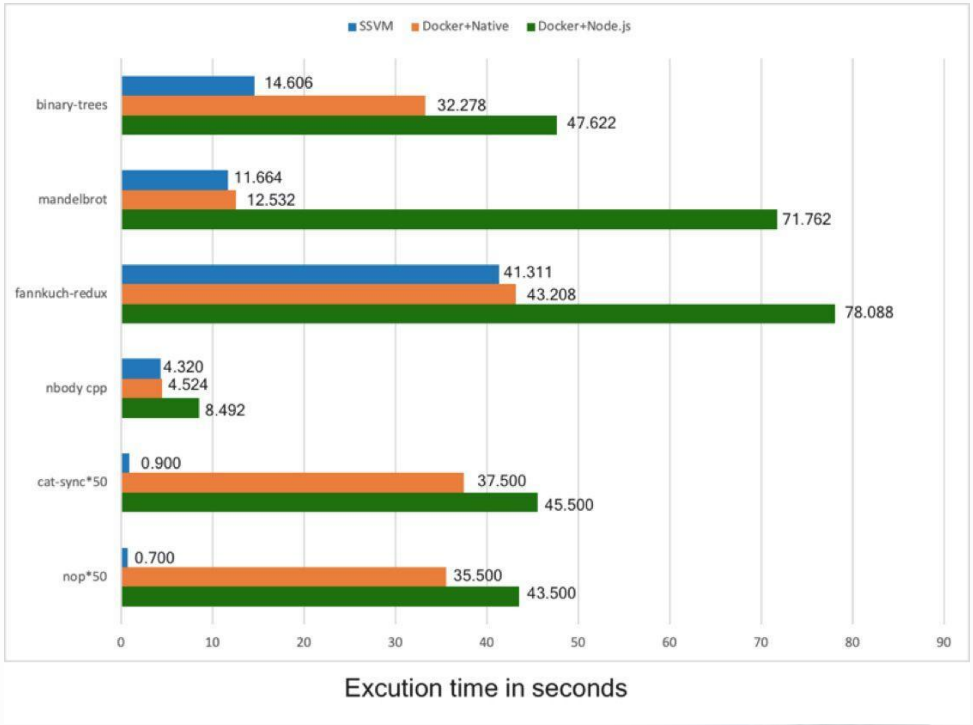
GOTC

Language agnostic

- Embed functions written in these languages into your Rust app
  - Rust
  - Tinygo (future Go)
  - C/C++
  - JavaScript
  - Python
- Great for creating plugins or extensions

Why WebAssembly (Wasm)?

Lighter and perhaps even faster!



Journals & Magazines > IEEE Software > Volume: 38 Issue: 1

A Lightweight Design for Serverless Function as a Service

Publisher: IEEE Cite This PDF

Ju Long ; Hung-Ying Tai ; Shen-Ta Hsieh ; Michael Juntao Yuan **All Authors**

4	1757
Paper Citations	Full Text Views

WasmEdgeRuntime

GOTC

A lightweight, secure, high-performance and extensible WebAssembly Runtime

- Support networking socket and web services
- Support databases, caches, and DOs
- Support AI inference in Tensorflow, OpenVino, PyTorch etc.
- Seamlessly integrates into the existing cloud-native infra
- Support writing wasm programs using JS

https://github.com/WasmEdge/WasmEdge

Challenges and solutions

Only supports generic CPUs

- Challenge:
  - Does not support advanced hardware features at the Wasm level
- Solution:
  - Use AOT to generate advanced CPU instructions
  - Supports new CPU features like SIMD
  - Use host functions to support hardware features like the GPU / TPU

[WASI-NN] Add GPU target support for pytorch backend. #2457

Merged hydai merged 2 commits into `WasmEdge:master` from `yanghaku:pytorch-cuda` 2 days ago

Conversation 6 | Commits 2 | Checks 54 | Files changed 2

yanghaku commented 3 weeks ago • edited by hydai — Contributor

This PR adds GPU target support for the WASI-NN Pytorch backend to speed up deep learning inference.

To use the GPU target, there are 3 requirements:

1. Hardware: Nvidia GPUs which support CUDA, AMD GPUs which support ROCM.
2. Runtime: `libtorch.so` must be built with CUDA/ROCM support.
3. User Code: Users should select the `GPU` target when loading the graph.

Fix #2140

Challenges and solutions

Limited system access

- Challenge:
  - WASI does not provide a complete set of POSIX features
  - Especially in the area of networking sockets and file system access
- Solution:
  - WasmEdge sockets allow non-blocking, DNS-enabled and TLS-enabled sockets
  - Rust
    - HTTP / HTTPS clients and servers: https://wasmedge.org/docs/category/44-http-services
    - Database drivers: https://wasmedge.org/docs/category/47-database-drivers
  - JavaScript
    - fetch() and node server: https://wasmedge.org/docs/develop/javascript/networking

Challenges and solutions

Single-threaded

- Challenge:
  - The guest app in the Wasm runtime cannot be multithreaded
- Solution:
  - The Wasm + WASI thread proposals to spawn new VMs in lieu of threads
  - Use co-routines on a single thread
    - Supports Rust's tokio framework
    - Future: the Wasm stack switching proposal (typed continuation)
    - Allows multiple concurrent network connections in a single VM instance: https://github.com/second-state/microservice-rust-mysql

Challenges and solutions

Single-threaded

- Challenge:
  - Rust tokio SDK does not have visibility into the binary-distributed Wasm instance
- Solution:
  - Fiber-based solution
  - The Wasm stack switching proposal (typed continuation to add pause, suspend, resume)
  - Async WASI implementation
  - Examples:
    - Async host functions: https://github.com/second-state/wasmedge-rustsdk-examples/tree/main/define-async-host-func
    - Async Wasm: https://github.com/L-jasmine/WasmEdge/tree/feat/async

Wasm will also make Rust better

High level of abstraction

- The Component model
  - Provides clearly defined modules with security boundaries
  - Allows Rust-like resource management (borrow checks) at runtime!
  - Provides tooling to generate APIs for many languages
- WASI NN for AI inference
  - Supports Tensorflow, PyTorch, and OpenVINO backends
  - Example: https://github.com/second-state/WasmEdge-WASINN-examples
  - Mediapipe support: https://github.com/WasmEdge/WasmEdge/issues/2355
  - Document AI support: https://github.com/sarrah-basta/wasmedge_ai_testing
- WASI Cloud: https://github.com/WebAssembly/WASI/issues/520

Wasm will also make Rust better

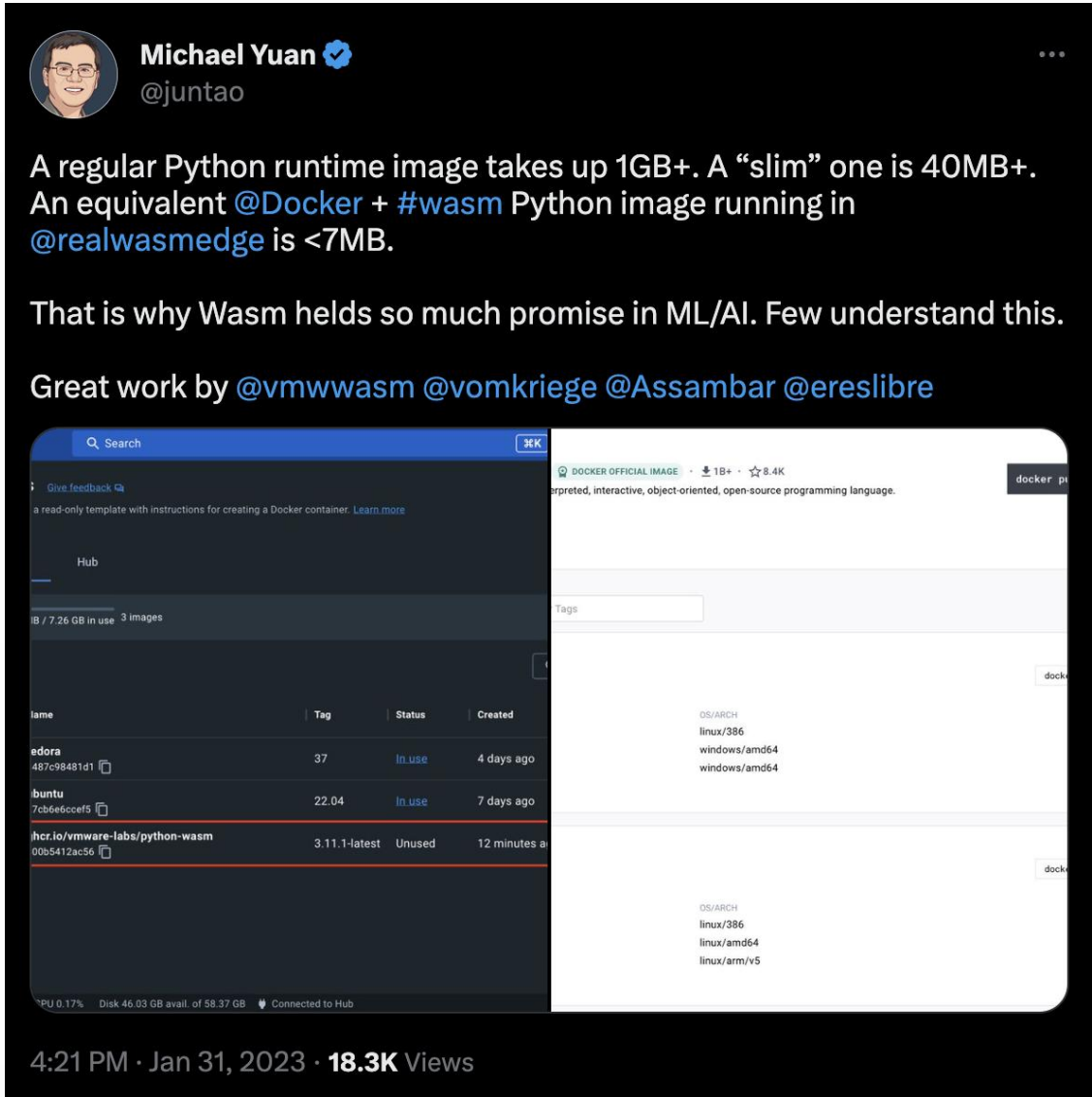
Enables Rust APIs for other popular languages

- JavaScript:

  https://wasmedge.org/docs/develop/javascript/rust

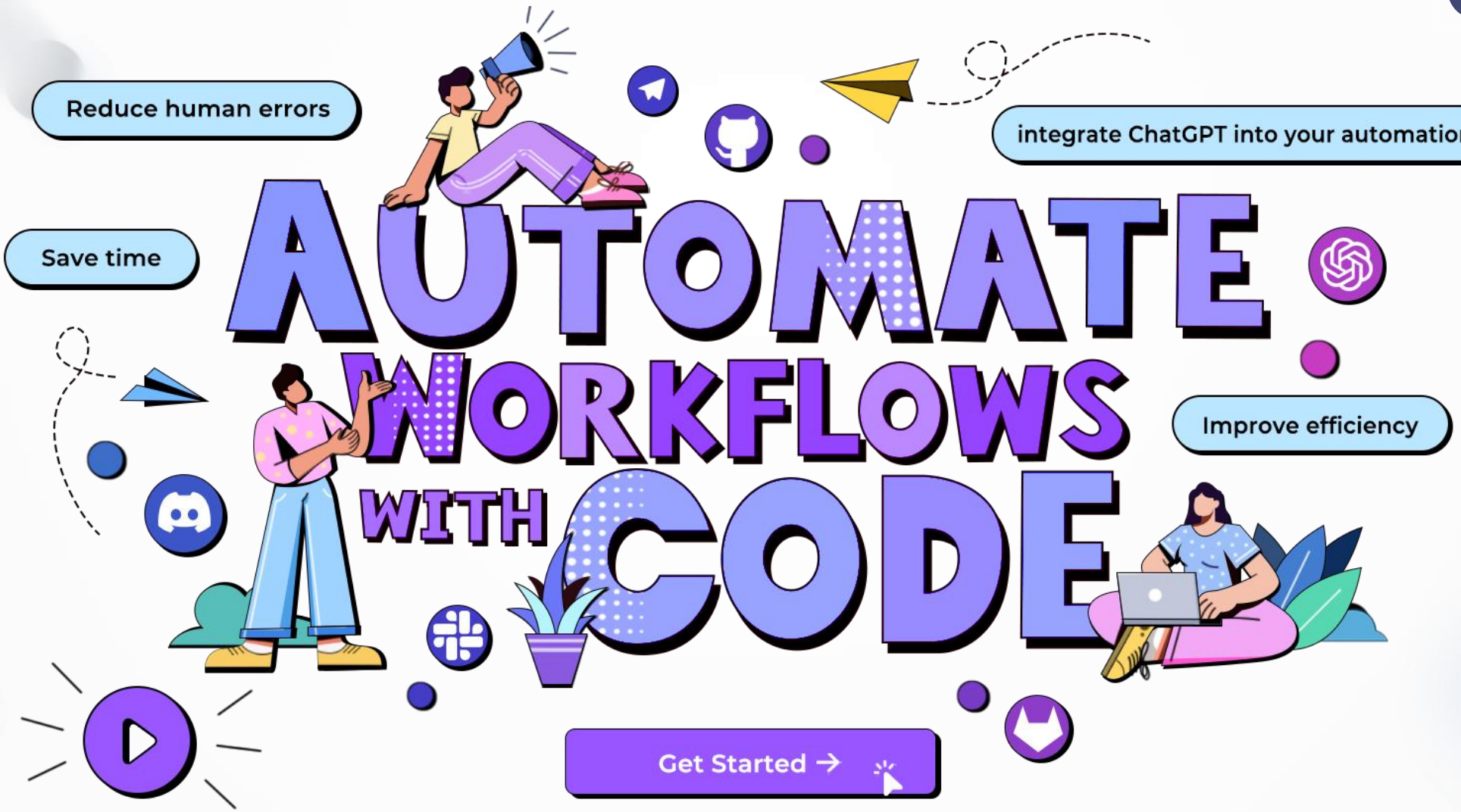
- Python:

  https://github.com/WasmEdge/WasmEdge/issues/2471



Example: Use Rust to create LLM plugins

Async, I/O intensive applications that must be easy to write



https://github.com/flows-network/

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

```
16 +     if n <= 1 {
17 +         return JsValue::Bool(false);
18 +     }
19 +     let limit = (n as f64).sqrt() as i32;
20 +     for a in 2..limit {
21 +         if n % a == 0 {
22 +             return JsValue::Bool(false);
23 +         }
24 +     }
25 +     JsValue::Bool(true)
26 + }
27 +
```

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.

```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.

```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.

```
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.

```rust
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.

GOTC

https://github.com/flows-network/github-pr-summary

1. **Load the code review bot template in flows.network.** The template contains the source code for the bot itself. We will clone the source code to your own GitHub account so that you can modify and customize it later. Click on Create and Deploy.
2. **Authorize bot access to GitHub.** The `github_owner` and `github_repo` point to the target GitHub repo where the bot will review PRs. Click on Authorize to give the repo the necessary permissions in GitHub.
3. **Give the bot your OpenAI API key.** If you have saved API keys in the past, you can skip this step and reuse these keys.

GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE