# RUQING YANG

yangrq.lambda@gmail.com github.com/waterlens

## INTERESTS

I aim to improve **programming languages** to enhance **performance** and provide stronger guarantees for users. Designing and implementing **optimizations** in compilers for programming languages has been my lifelong pursuit.

## EDUCATION

**Hong Kong University of Science & Technology** **Sept. 2023 - Nov. 2025 (expected)**
**M. Phil.** in *Computer Science and Engineering*. Supervised by Lionel Parreaux. *Hong Kong S.A.R., China*
Research track: **optimizations** for functional programming languages.

**Zhejiang University** **Sept. 2019 - June 2023**
**B. Eng.** in *Computer Science and Technology*. GPA: 3.84/4.0 *Hangzhou, China*

## PROJECTS

**Calocom** 🔗 ***Rust*** **Spring 2022**

- This is a group project for course *Compilation Principles*.
- A feature-rich programming language with **algebraic data types**, high order functions, and pattern matching.
- I was involved in designing the **type system**, typed AST, the memory representation of objects, the style of name mangling, and a middle IR that provides an intermediary level for desugaring and other necessary transformations.
- I led the development of this project. I implemented almost all components (except for lexing & parsing), including syntax desugaring, **semantics checking**, **closure conversion**, **LLVM**-based **code lowering** with Rust library *inkwell*. I also wrote standard libraries (strings, vectors, etc.) and **runtime** (objects allocation and program entry point) in unsafe Rust.
- Best course project in my class.

**SyOC** 🔗 ***C++, Python, ARM*** **Spring 2022 - Summer 2022**

- A hobby project with my friend for learning compiler optimization techniques and participating in Bisheng Cup Compiler Contest. We wrote the project from scratch and were the first team (2 people) from ZJU to enter the final round of the contest.
- I led the development of this project. I designed the basic **compiler framework** for performing optimization transformations with C++ templates, and a **SSA**-based intermediate representation with **def-use** and **use-def** chains.
- I implemented the lexer, the recursive descent parser, the **mem2reg** pass (including **immediate dominator analysis** and **iterated domination frontier analysis** for SSA construction), dead code elimination, and constant propagation.
- I wrote a Python script for comparing the performance of the optimized program with gcc or other compilers.

**QuicKaml** 🔗 ***C*** **Autumn 2023**

- A hobby project that implements a **register-based VM** interpreter of a monomorphic language and engineered many low-level optimizations.
- I patched LLVM with special **calling conventions** to generate efficient code for the handler of VM instructions in interpreter using **guaranteed tail-calls**.
- I tried multiple techniques to improve the performance of the interpreter, including: **operands reordering** to allow more efficient sign-extension; **partial register decoding** to reduce unnecessary shifts on x86-64 architecture; instruction fetching based on unaligned memory access or shifting & masking; **pre-decoding** before dispatching to exploit the CPU pipelines.

**MLscript** 🔗 ***Scala, C++*** **Autumn 2023 - Now**

- A joint project from my lab.
- I designed a **ANF**-based intermediate representation with join points extension.
- I implemented a **smart inliner** with control flow analysis to identify when to make inlining decisions, and leverage **function splitting** technique to minimize the code duplication brought by inlining. I am also responsible for the implementation of the C++ backend, which features a universal memory representation for objects, **optimized arithmetic operations on unboxed values**, and **reference-counting**-based memory management.

**MMM** 🔗 ***MoonBit, RISC-V, WebAssembly*** **Autumn 2024**

- A joint project with my friend for the MGPIC contest. Won 1st place and had an absolute advantage over 2nd place.
- I led the development and designed an optimizing compiler framework in MoonBit for the *Mini MoonBit* language with JS, **RISC-V** and **WASM** backends.
- I implemented all essential optimizations for the contest, including **guaranteed tail recursion elimination**, **selective lambda lifting** based on **register pressure**, **basic block straightening**, dead code elimination, **local value numbering**, **common subexpression elimination**, **loop invariants code motion**, **jump table optimizations**, **scalar replacement**, and fast bump allocating.
- I wrote the code generators for JavaScript and RISC-V backend. To avoid stack overflow, I devised a **selective CPS transformation** and **automatic thunking** on function calls in the JavaScript backend. In the RISC-V backend, I ported a **tree-pattern covering instruction selector** from Cranelift, and implemented a **chordal graph coloring register allocator**.
- I extended the language with parametric polymorphism (**generics**), ad-hoc polymorphism (**typeclass**, implemented through dictionary-passing), and user-defined operators.

**RMatch** 🔗 ***C++*** **Autumn 2021**

- A personal hobby project that parses **regular expressions** and generates NFA-based **virtual machine** bytecode. The bytecode is then **JIT**-compiled to native x86-64 machine instructions with C++ library *xbyak*.

**Apple μArch Bench** 🔗 ***C*** **Spring 2024**

- A hobby project to explore **micro-architecture** characteristics on Apple Silicon with **hardware performance counters**.

**SIB Optimization for OCaml** 🔗 ***OCaml*** **Spring 2025**

- *Share-immutable-block* optimization. Functional programming languages frequently perform pattern matching on existing data structures. Even if the new object created is identical to the old one, a new object is often allocated. I implemented a sound optimization that eliminates this unnecessary allocation if the object is proven to be immutable.
- This optimization is internally used in the MoonBit compiler.

**Monoid Hash** 🔗 ***C, AArch64*** **Spring 2025**

- The performance critical part of an ongoing research project on incremental computation.
- I extended the fast-crc32 implementation with hardware-accelerated monoid combination using ARMv8's `pmull` instructions. Specifically, this acceleration involves speeding up the multiplication of two bit-reflected polynomials over the $\mathrm{GF}(2^{32})$ field.

## PUBLICATIONS

**Smart Inlining through Function Splitting**, ***PLDI SRC 2025*** **April 2025**

## EXPERIENCE

**Intern for Programming Language Tool Development**, *at IDEA* **Mar. 2025 - Sept. 2025 (expected)**

- I implemented an OCaml optimization that improves the performance of the MoonBit compiler.
- I improved the speed of compiling the MoonBit test when using the native backend by using the `tcc` compiler to compile generated C source, which involved a refactoring of build system, fixing bugs in `tcc` compiler, and cross-platform (Linux, macOS, Windows) support for runtime library of MoonBit.
- I added the benchmark feature to the toolchain with statistical analysis and visualization.

**Student Volunteer**, *ICFP 2024* **Sept. 2024**

**Teaching Assistant**, *Programming with C++* **Jan. 2024 - June 2024**

- I designed a lab that helps students to understand the pointer and reference in C++.

**Remote Research Intern**, *hosted by Yizhou Zhang* **Sept. 2022 - Jan. 2023**

- I studied the implementation and semantics of **lexical algebraic effects**, which is a hot topic in programming language research.

**Undergraduate Teaching Assistant**, *Principles of Programming Languages* **Sept. 2022 - Jan. 2023**

- I designed a lab in OCaml that helps students to understand Hindley-Milner **type inference** algorithm.
- I set a homework to assist students to learn and use the **evaluation-context-style operational semantics** and **delimited continuation**.
- I designed and implemented an online judge system for the course, utilizing public GitHub repositories and free CI (GitHub Actions) quotas. To ensure the privacy of the students' code, I devised an approach to require students use a public key to encrypt their code before submitting their code as a GitHub issue.

## SKILLS

**Programming Languages:** Proficient in multiple programming languages, including but not limited to:

- Most frequently used: OCaml, Rust, C/C++, Scala
- Familiar: Java, Python
- Experienced with: TypeScript, JavaScript, Ruby, Haskell, Lua, Verilog, Scheme, etc.

**Programming Language Theory:**

- Formal verification with Coq.
- Read books on programming languages, including: **Software Foundations**; **Types and Programming Languages**; **Practical Foundations for Programming Languages**; **Essentials of Programming Languages**.
- Constraint-based type inference, bidirectional type inference, etc. Rich knowledge on type system.

**Compilers:**

- Experienced in using and modifying common compiler frameworks, such as LLVM, Cranelift, etc.
- Familiar with the compilation of various paradigms of programming languages, including imperative, functional, object-oriented, and dynamic languages.
- Skilled in manually tuning micro-architecture-level performance using **profiling** tools such as `perf`, `VTune`, and `flamegraph`.
- Understanding of multiple **register allocation** algorithms (iterated register coalescing, linear scan, etc.), **garbage collection** algorithms (mark-sweep, mark-compact, tri-color incremental, generational, etc.).
- Extensive knowledge of interpreter and runtime system design and implementation, including various threading techniques, stack-based VM and register-based VM, memory management, runtime objects representation, **context switching**, etc.

**Languages:**

- Chinese (native), English (good working communication)