中山大學 SUN YAT-SEN UNIVERSITY
计算机学院（软件学院）
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

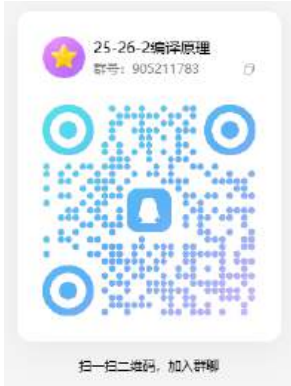
Compilation Principle
编　译　原　理

第1讲：概述

张献伟

yatcc-ai.com

DCS290, 3/4/2026



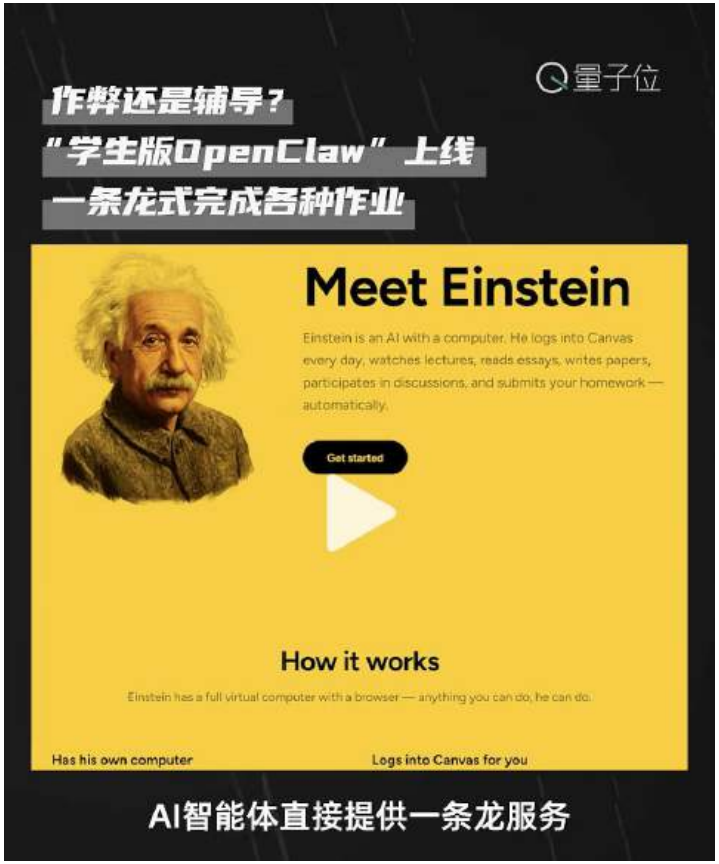
YAT COMPILER

ARISYSU

Impact of LLM

大模型编程给计算机专业学生带来的冲击是什么

大模型编程（如基于GPT类模型的代码生成、调试与重构）对计算机专业学生带来的冲击，是**结构性、长期性、范式级的改变**。它既不是简单的“工具升级”，也不是单纯的“效率提升”，而是对能力结构、学习方式与职业路径的重塑。



总体判断

大模型对计算机学生的冲击不是“替代”，而是**重新分层**：

- 低水平编码能力会被压缩
- 中等能力学生面临竞争压力
- 高水平学生会获得能力放大器

如果学生把大模型当“自动写代码工具”，会变弱；
如果把它当“思维增强器与系统协作工具”，会变强。

一句话总结

> 大模型时代，写代码不再稀缺，
> 真正稀缺的是理解复杂系统、控制抽象层次和驾驭智能工具的人。

Words of Chris, Again

> *AI coding is therefore best understood as another step forward in automation. It dramatically lowers the cost of implementation, translation, and refinement. As those costs fall, the scarce resource shifts upward: deciding what systems should exist and how software should evolve.*

Closing Thoughts

The Claude C Compiler doesn't mark the end of software or compiler engineering. If anything, it opens the door wider. The easier implementation gets, the more room there is for genuine innovation.

Lower barriers to implementation do not reduce the importance of engineers; instead, they elevate the importance of vision, judgment, and taste. When creation becomes easier, deciding *what is worth creating* becomes the harder problem. AI accelerates execution, but meaning, direction, and responsibility remain fundamentally human.

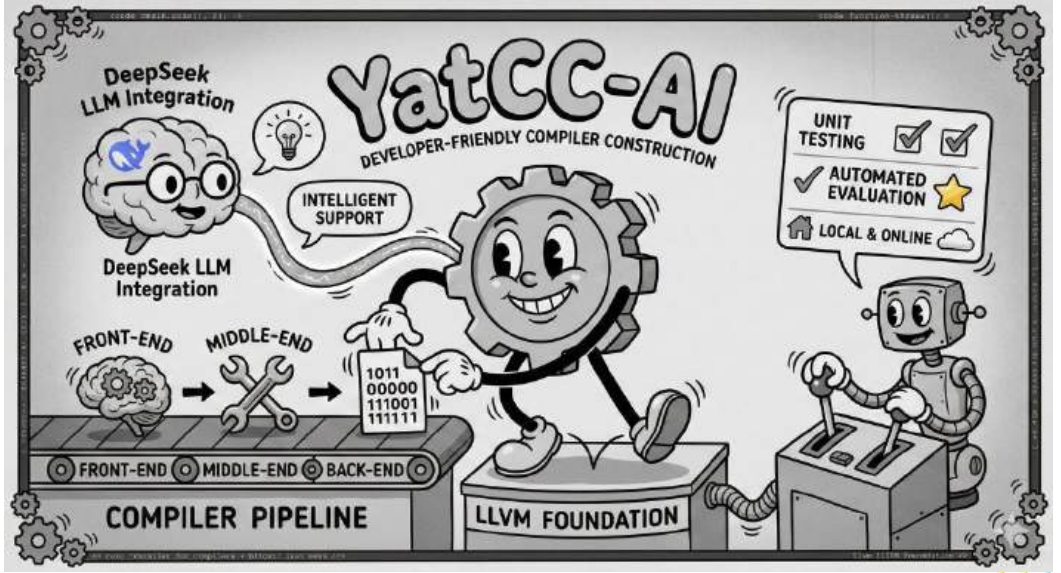
Writing code has never been the goal. Building meaningful software is. The future belongs to teams willing to embrace new tools, challenge assumptions, and design systems that help people create together.

That is the future that has driven Modular's mission from the start, and the one I believe this new era of AI makes possible.

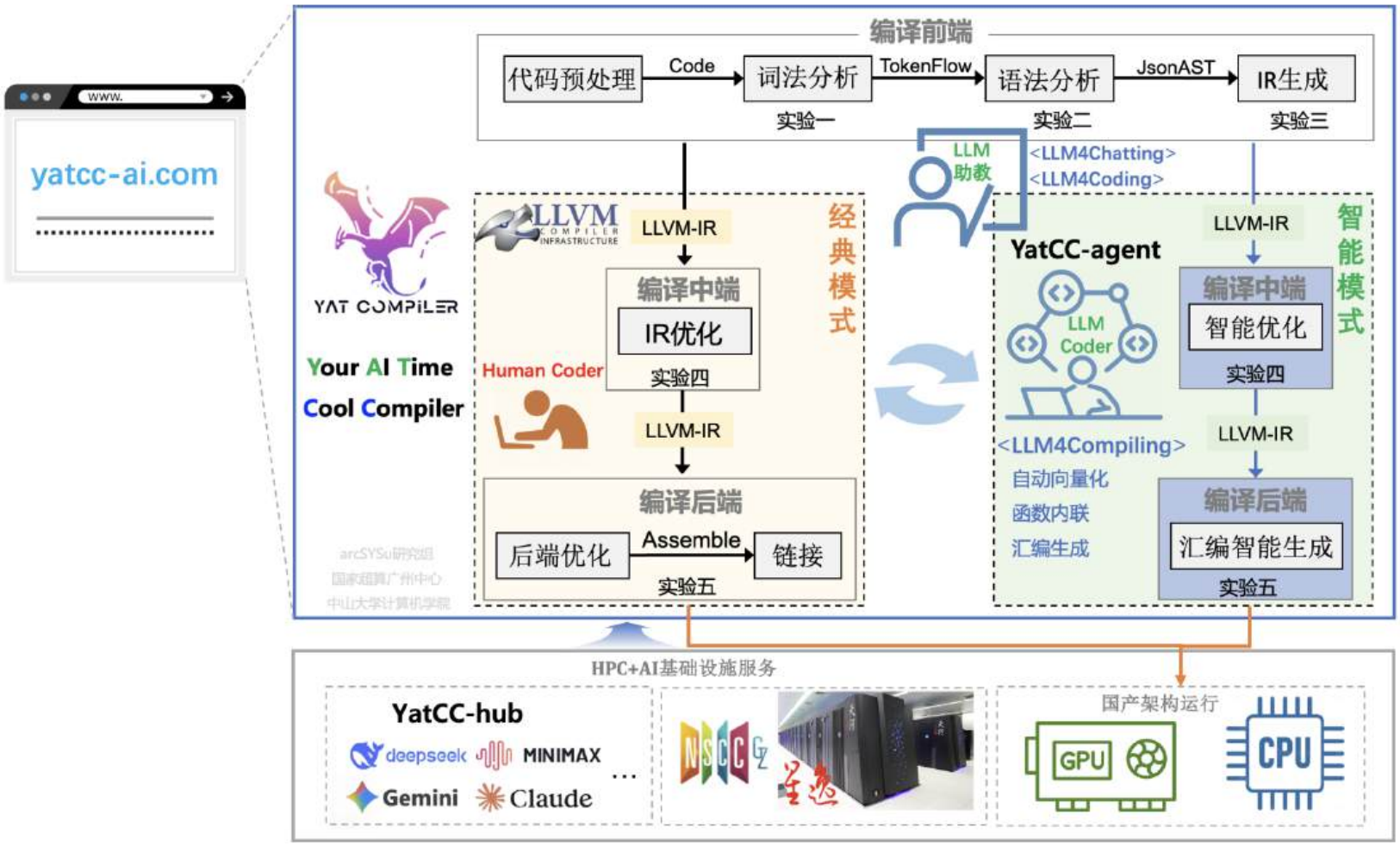
- Chris Lattner

https://www.modular.com/blog/the-claude-c-compiler-what-it-reveals-about-the-future-of-software

YatCC-AI[yatcc-ai.com]



YatCC-AI[yatcc-ai.com]



YatCC是什么？

- **起源：中大编译课程**
  - Yat Compiler Course
- **发展：智能实践平台**
  - Your AI Time Cool Compiler
- **长期：纵向自主项目**
  - Yat Creative Cloud
  - 首个长期的arcSYSu组内项目
  - 已经积累了巨大投入
- **影响：一张对外名片**
  - 超算中心 → 计算机学院 → 中山大学

YatCC发展历程

吴坎
2021硕
/2023博

2022之前
Yat-Sen OS
YatCPU

2022
SYsU-lang

顾宇浩
2022硕
/2024博

2023
SYsU-lang

郑腾扬
2024硕

2024
SYsU-lang2

YAT COMPILER

2025
YatCC

YatCC整体架构

- **逻辑结构**

接口形式：

- Web Page

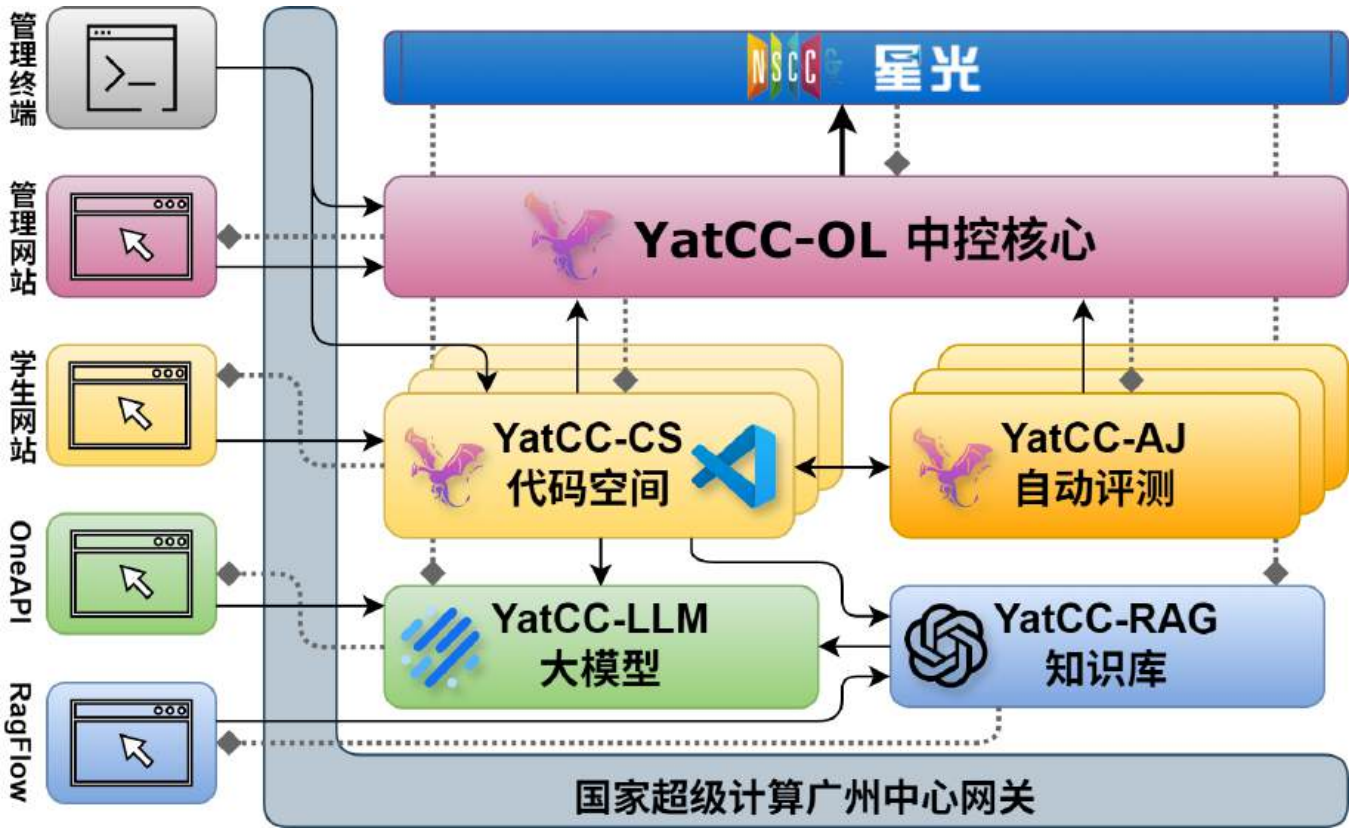
人类用户界面

- RESTful API

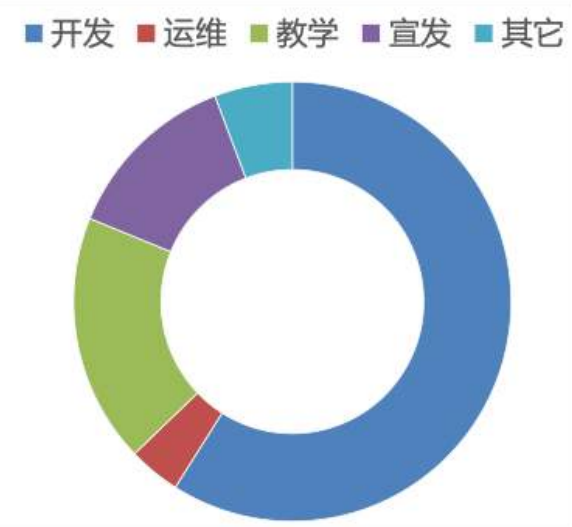
系统操作

- SSH

调试运维



人员投入（2025）



相关报道

DeepSeek+超算：中山大学YatCC-AI平台赋能编译实践课程建设，您加入吗？

中山大学教务部 2025年03月20日 22:20 广东

1人 星标

一、 中山大学YatCC-AI平台是什么？如何用于编译实践课程建设？

随着以DeepSeek大模型为代表的人工智能（AI）技术和以超算为代表的高性能计算（HPC）技术的飞速发展，计算机领域正经历着前所未有的深度变革。如何将这些前沿技术融入课程教学，培养适应未来智能时代需求的创新型人才，成为教育界亟待解决的关键问题。在此背景下，中山大学计算机

中山大学教务部 23 89 9 写留言

G.O.S.S.I.P 资源推荐 2025-04-18 编译器的开发之道

原创 G.O.S.S.I.P 安全研究GoSSIP

2025年04月18日 22:12 上海 2人 星标

我们在本月的早些时候介绍了关于使用编译器安全选项的文章（《G.O.S.S.I.P 阅读推荐 2025-04-09 编译器的安全之道》），那么今天要带大家去看一个很有趣的编译器开发教程：

学过编译原理的同学应该都知道著名的Yet Another Compiler Compiler也就是YACC工具，它是一个经典的生成语法分析器的工具。而大家也知道广州有一所刚刚在去年度过百岁生日的知名大学——中山大学，当然，千万不要把Sun Yat-sen University弄成是"双鸭山大学"哦。那么这两者的结合会碰撞出什么火花呢？那就是我们今

欧阳易凡

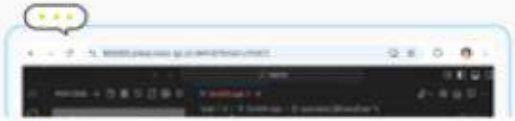
安全研究GoSSIP 27 68 7 写留言

YatCC-AI：超算+DeepSeek，中山大学开启"智能编译"实践教学

中山大学计算机学院 2025年02月25日 12:09

广东 听全文

近期，DeepSeek在全球技术领域掀起浪潮，推动国产算力产业蓬勃发展，迅速引发各领域的深度变革。如何将这些前沿技术有效融入课程教学，培养面向未来智能时代的专业人才，成为教育领域的重要课题。对此，计算机学院张献伟副教授携手超算中心团队率先给出了思考和创新方案，成功研发并上线了基于**国产超算**和**DeepSeek**的**YatCC-AI**智能编译教学实践平台（https://yatcc-ai.com）。该平台将高性能计算（HPC）和AI技术深度融入计算机系统课程，为2025学年核心课程《编译原理》与《编译器构造实验》提供了智能化、现代化的全新学习体验。



中山大学计算机... 60 112 20 写留言

荣誉奖项


yatcc-ai.com
基于国产超算和大模型的智能编译实践
教育部教师人工智能应用培育案例国家级推荐
广东省优秀教学成果（本科类）特等奖
广东省优秀教学成果（研究生类）一等奖
中国计算机教育大会CECC优秀案例特等奖
中国计算机教育大会CCEC优秀案例特等奖
中山大学本科教学成果特等奖
中山大学教学竞赛二等奖
中山大学研究生教学成果特等奖

思考

2021 年 4 月，香山团队爆发了一次“信心危机”。团队一些核心骨干开始怀疑香山项目，觉得和工业界相比香山就是“垃圾”。学生们也开始产生怀疑，从香山的技术方案到老师们的指导，再到怀疑整个项目的意义。我们连开了好几次会，确定了若干条总原则，比如强化“先完成、后完美”理念，遵循“不空口争论、用数据说话”，积极与业界专家交流获取反馈等等。彼时，我也和当时几位项目骨干单独沟通，鼓励和激励他们坚定信念。

https://www.bilibili.com/opus/674485155974348800

❖ 最根本的意义是什么?

- **就是项目本身**
  - 一个人的力量很渺小
  - 大家在一起就能做成很多事
    - 项目像一面旗帜，将大家收拢到一起

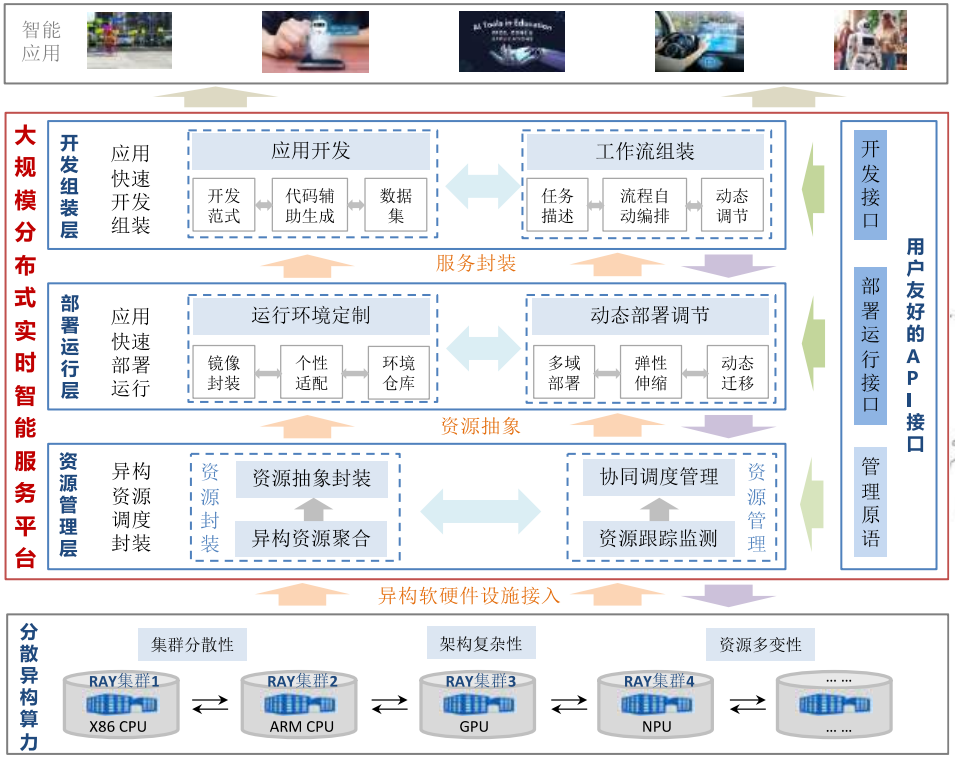
❖ 核心竞争力是什么?

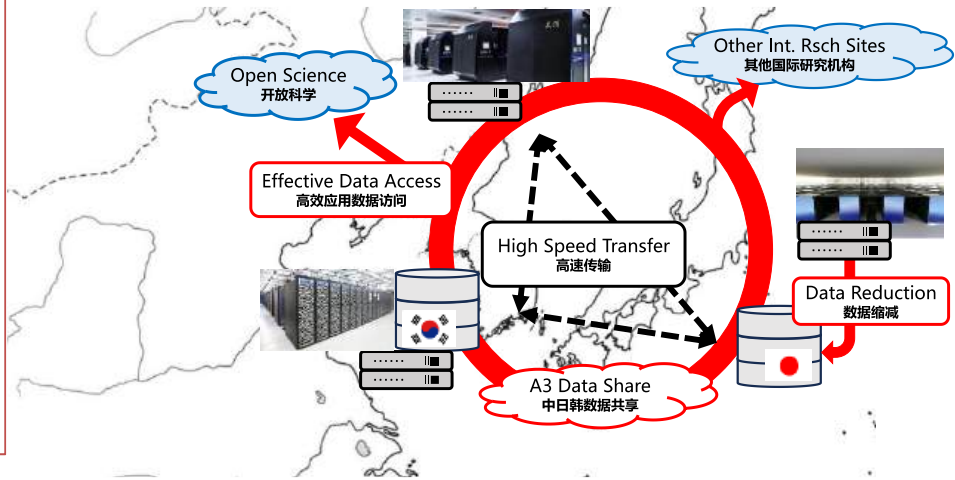
- **优质课程**
  - 生态不能指望别人帮你建设
- **自主闭环**
  - 年年换新的可控学生用户群体
- **开放社区**

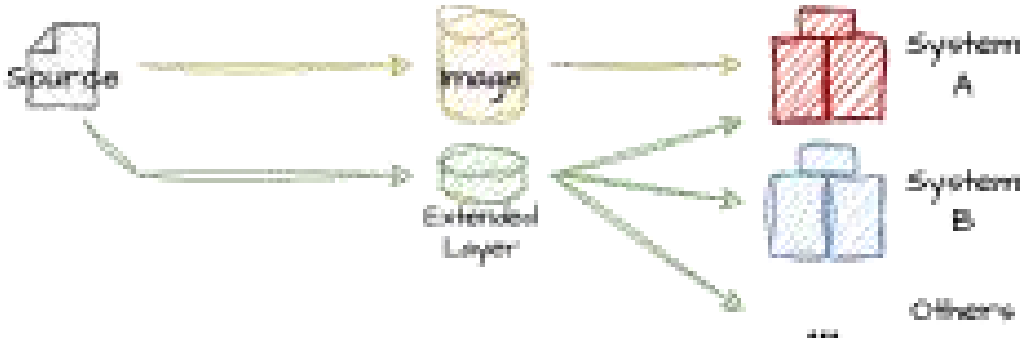
• 未来的机会在哪里?

- **技术试验田**
  - 科研联系实际的极好场景
- **产品孵化器**
  - “面向超算互联网的通用云开发平台前置技术研究”

科研关联





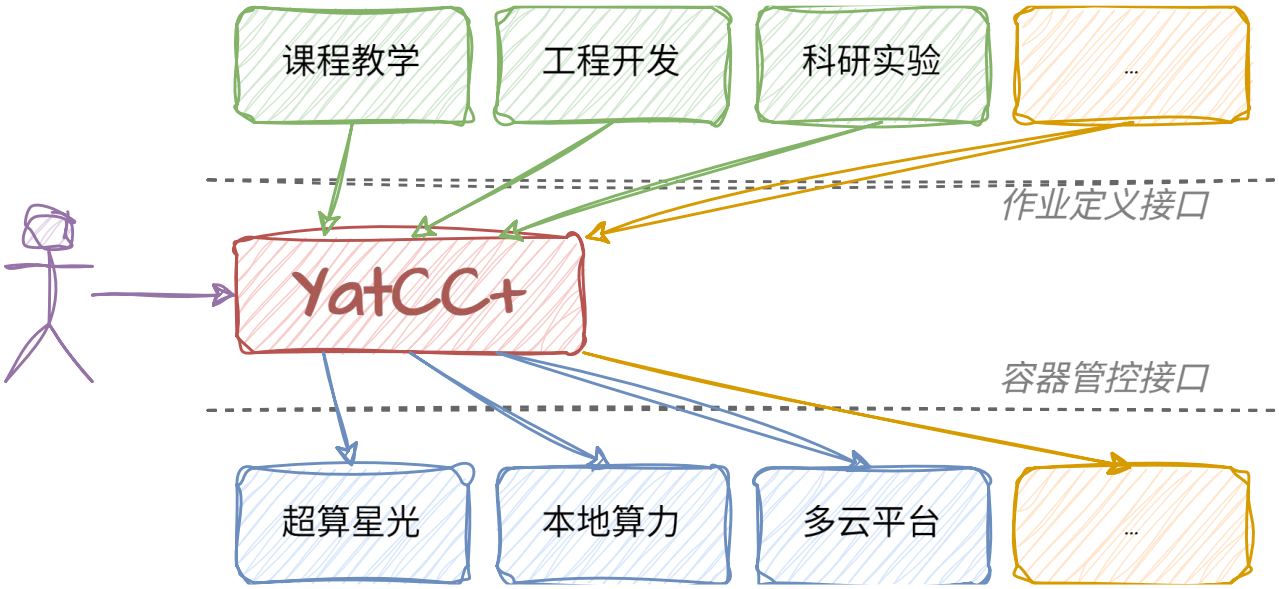


SC25
St. Louis, MO
hpc ignites.

YatCC定位

设施化、从用户出发：YatCC-AI → YatCC（Yat Creative Cloud/创逸云）

- 编译：edu.yatcc-ai.com/compiler
- 教学：edu.yatcc-ai.com
- 科研：sci.yatcc-ai.com
- 专业：pro.yatcc-ai.com



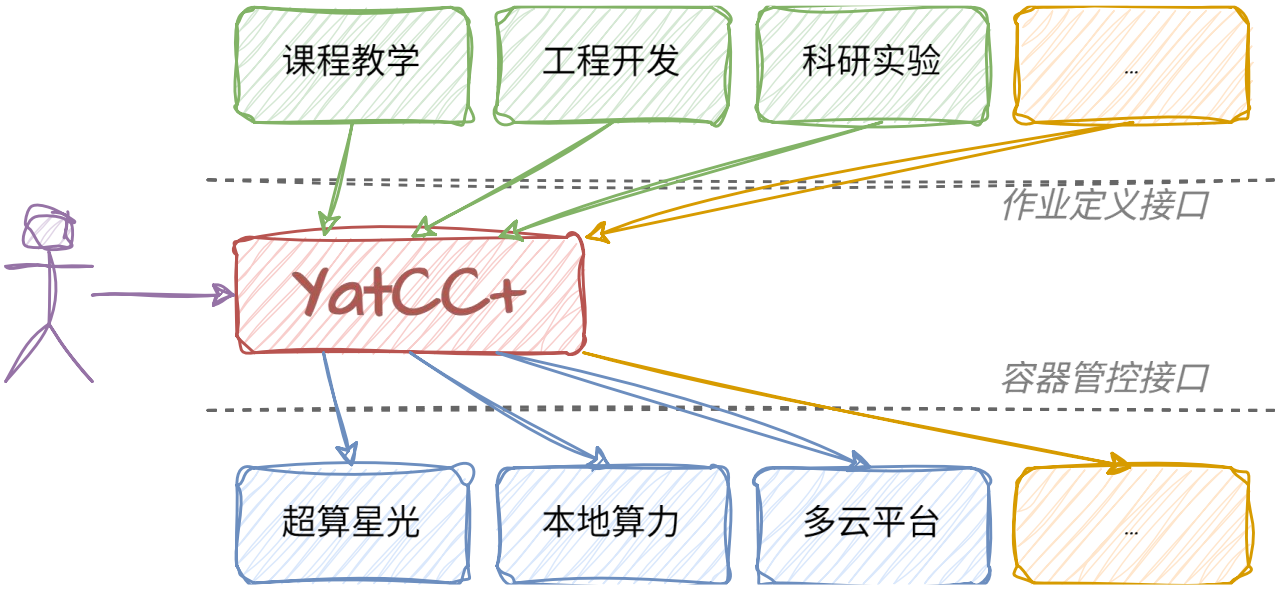
统合组内资源、首要服务自身、快速打造闭环

YatCC定位

设施化、从用户出发：YatCC-AI → YatCC（Yat Creative Cloud/创逸云）

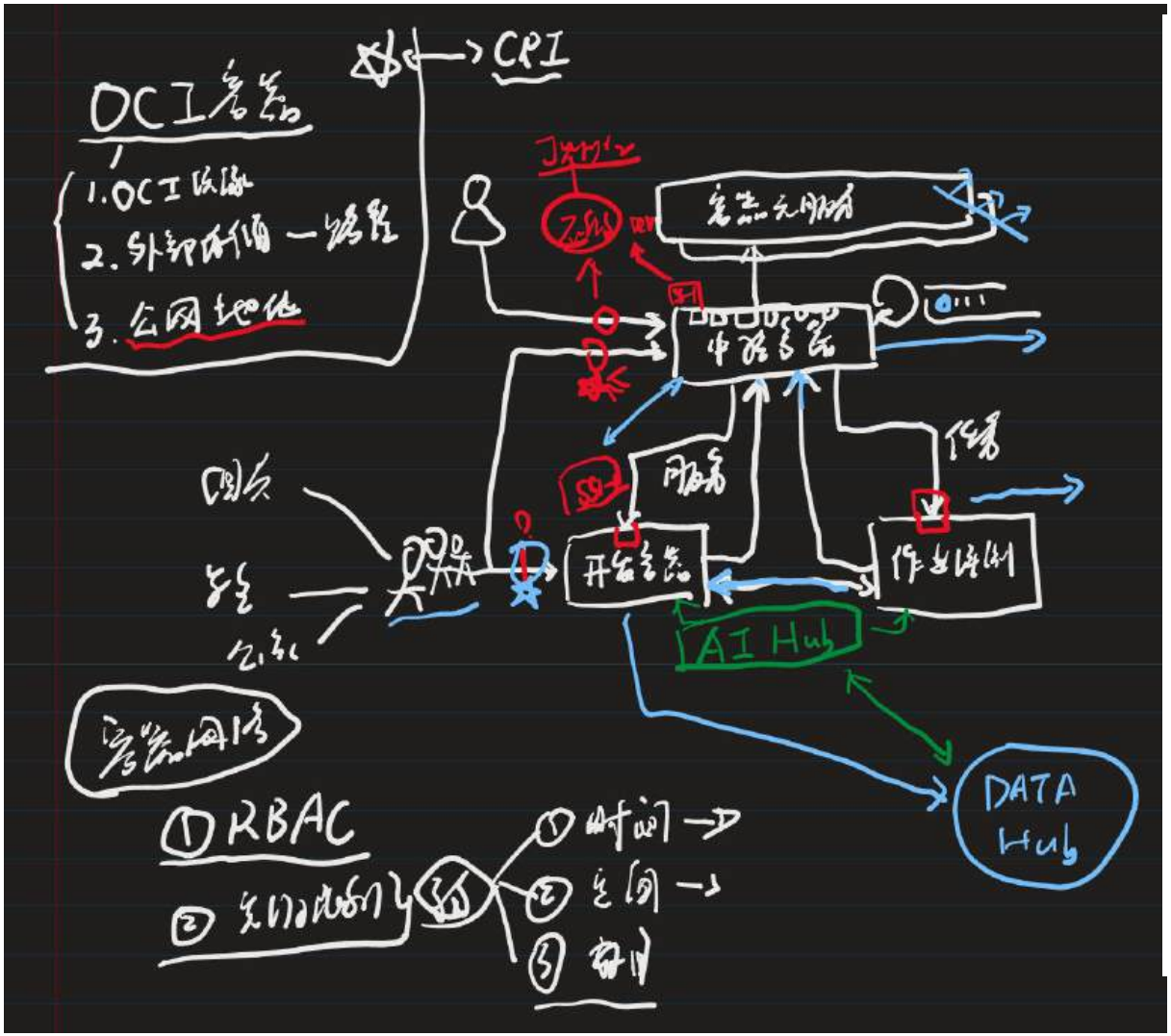
- 编译：edu.yatcc-ai.com/compiler
- 教学：edu.yatcc-ai.com
- 科研：sci.yatcc-ai.com
- 专业：pro.yatcc-ai.com

自己即用户



统合组内资源、首要服务自身、快速打造闭环

YatCC升级重构



端到端流程：

1. 用户通过 **Portal / API** 提交请求
2. 进入 **统一接入网关**
3. 调用 **认证与授权系统（OC / RBAC）**
4. 合法请求进入 **中央调度与编排系统**
5. 调度系统：
   - 选择 **计算资源**
   - 选择 **数据资源**
   - 调用 **AI Hub**
6. 任务在 **执行平面** 运行
7. 执行过程中：
   - 状态回传
   - 监控与日志采集
8. 结果返回用户
9. 同步进行 **审计 / 计量 / 计费**

YatCC for Agent

自然过渡

- Past: 面向教学的智能化代码开发、调试、运行的一站式一体化服务平台
- Now: 面向科研实践的智能化开发、部署、运行的一站式一体化服务平台
- Future: 面向LLM智能体全生命周期的一站式一体化服务平台

目标：构建一个覆盖 开发、训练、评测、部署、运营、治理 的LLM智能体一体化平台

- 智能体“低门槛构建”
- 工具与知识“可插拔集成”
- 运行资源“弹性调度”
- 安全合规“内生保障”
- 效果优化“持续闭环迭代”

- 智能体的开发环境
- 算力调度平台
- 工具生态系统
- 数据与知识中枢
- 运营与治理平台

YatCC: 过去 → 现在 → 未来

Yet Another Tiny C Compiler

Your AI Time Creative Cloud

Your Agent Creative Center

Automatic

AI

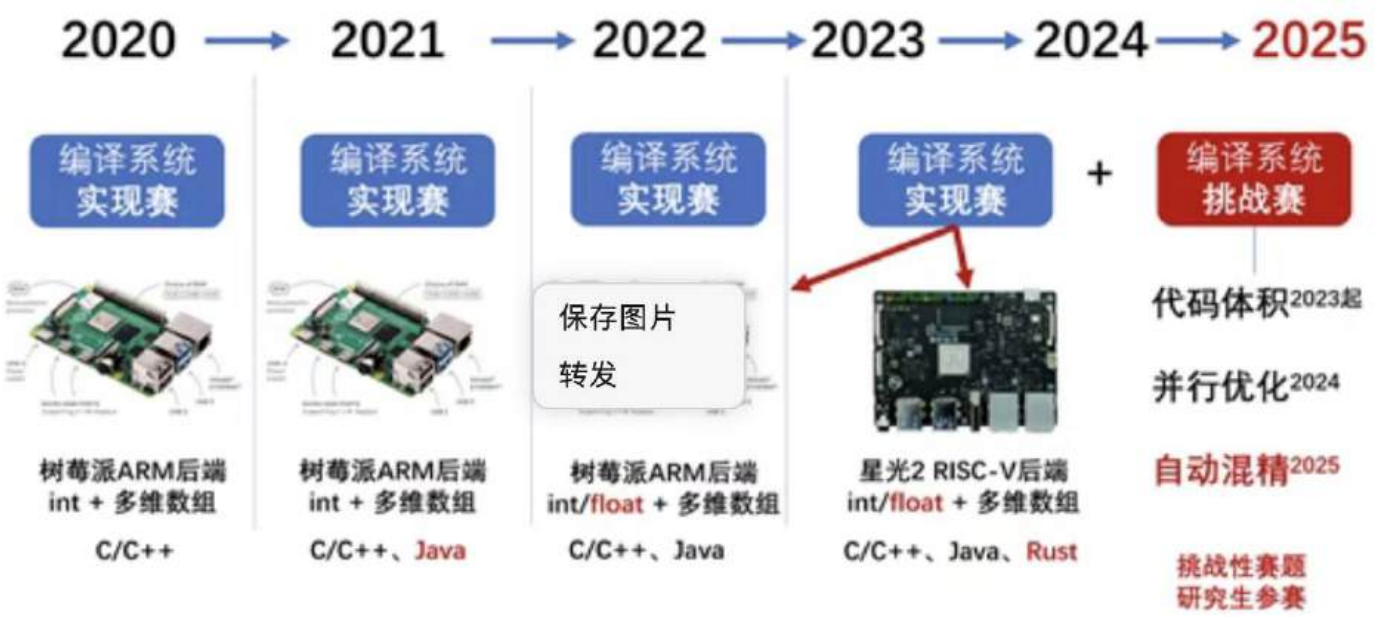
Agent

从一个综合视角去观察、感受、引领AI/LLM/Agent所带来的变革

Do the right thing, at the right time, with the right people

Be Professional, Do Smart, Show Impact

编译比赛



获奖证书

Yat-CC:

在2023年全国大学生计算机系统能力大赛编译系统设计赛中，荣获

一等奖

参赛单位：中山大学
队伍编号：202310558201558
参赛队员：郑中淳 黄瀚 潘文轩 蒋冠一

大赛技术委员会主任

全国高等学校计算机教育研究会
全国大学生计算机系统能力大赛组委会
二〇二三年八月二十三日

编译比赛(cont.)

2026编译大赛-挑战赛题丰富

编译器方向

- 基于AscendNPU IR的NPU自动融合代码生成与调优
- 基于Triton-Ascend构建昇腾亲和算子
- 动态算子图编译与并行调度
- 基于进化算法的Triton自动优化系统
- 并行优化
- 代码自动混精
- 代码体积优化

仓颉语言方向

- 仓颉高性能Range Analysis分析
- 仓颉数据竞争静态检测
- 仓颉代码片段语义检查

语言虚拟机方向

- Java虚拟线程实现机制优化

编译比赛 (cont.)

- 2024开始，推免资格加分

《计算机学院推免资格认定遴选细则》（征求意见稿）修订内容说明：

更新细则中的比赛清单，增加中国“互联网+”大学生创新创业大赛、全国大学生计算机系统能力大赛、国际大学生超算竞赛（SC)、国际超算学生集群竞赛（ISC) 等4个比赛，其他内容不变。

中国“互联网+”大学生创新创业大赛	国家级	金奖	80
		银奖	20
全国大学生计算机系统能力大赛	国家级	特等奖	80
		一等奖	80
		二等奖	20
国际大学生超算竞赛（SC)（线上、线下）	国际级	冠军、亚军、季军	80
国际超算学生集群竞赛（ISC)（线上、线下）	国际级	冠军、亚军、季军	80

未有上述类型获奖者，本项记为 0 分。

权重值：0.02。

中山大學 SUN YAT-SEN UNIVERSITY
计算机学院（软件学院）
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编　译　原　理

第1讲：词法分析（1）

张献伟

yatcc-ai.com

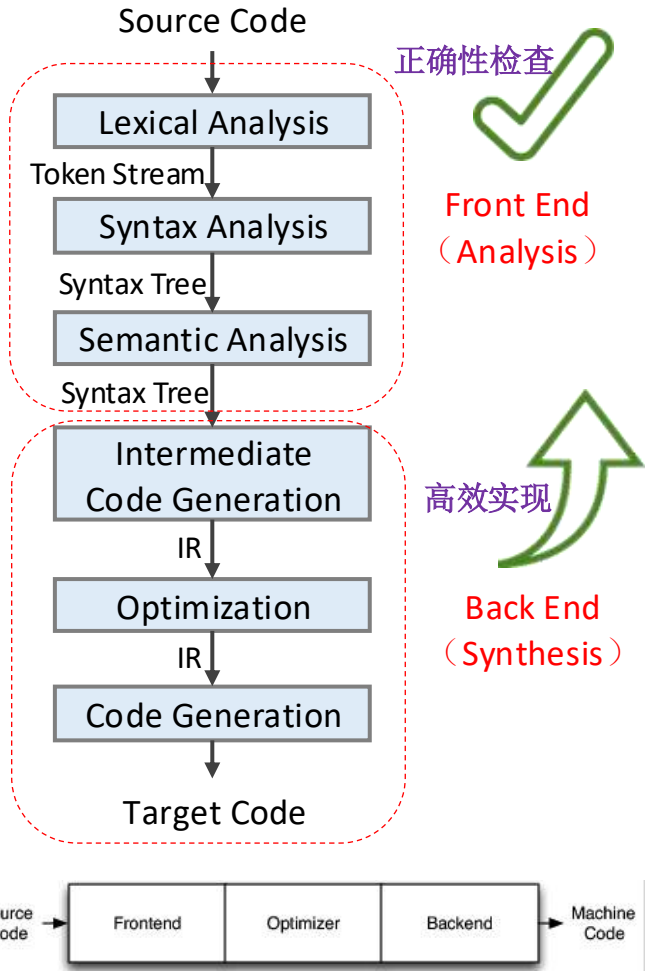
DCS290, 3/4/2026

YAT COMPILER

ARISYSU

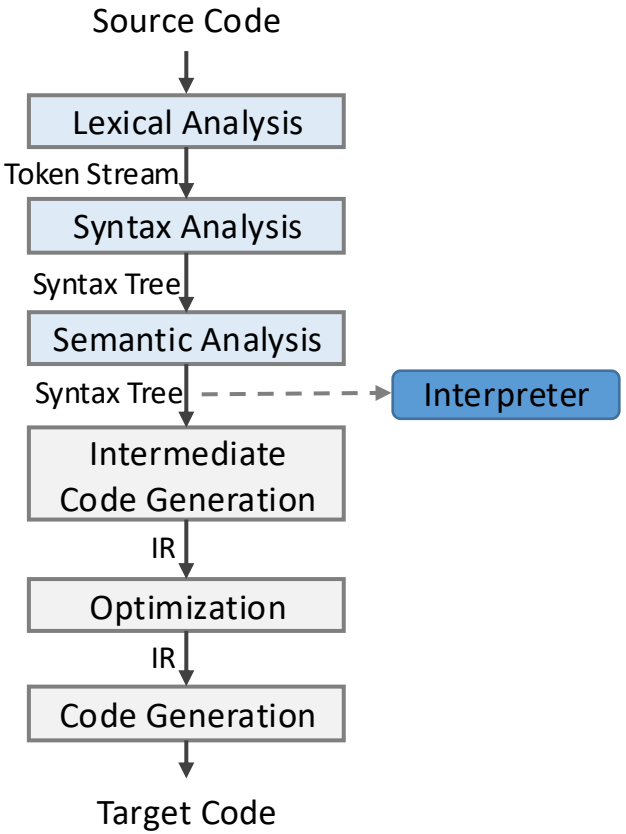
Compilation Procedure[编译过程]

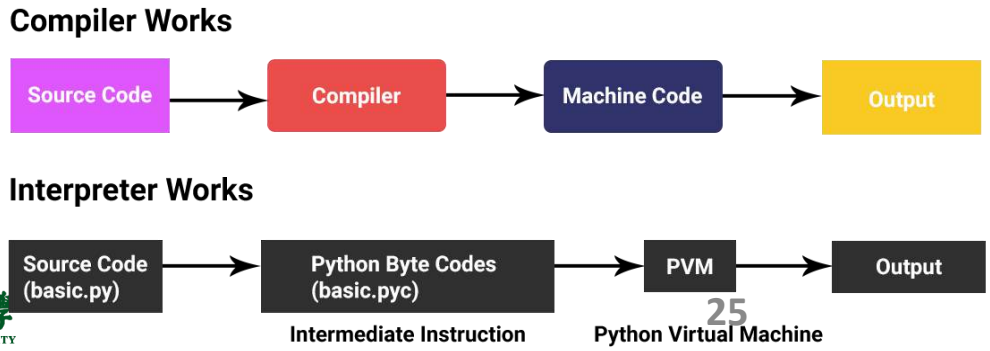
- **前端**（分析）：对源程序，识别语法结构信息，理解语义信息，反馈出错信息
  - 词法分析（Lexical Analysis） 词
  - 语法分析（Syntax Analysis） 语句
  - 语义分析（Semantic Analysis） 上下文
- **后端**（综合）：综合分析结果，生成语义上等价于源程序的目标程序
  - 中间代码生成（Intermediate Code Generation）
    - Intermediate representation (IR) 转换
  - 代码优化（Code Optimization） 更好
  - 目标代码生成（Code Generation） 可执行



Interpret vs Compile[解释 vs. 编译]

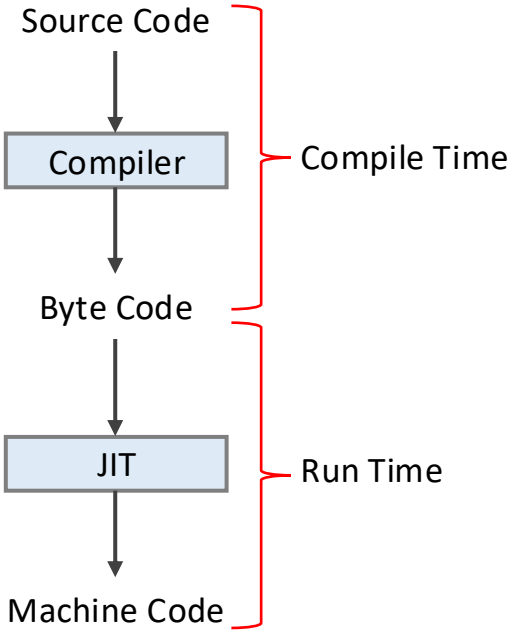
- **编译**：翻译成机器语言后方能运行
  - 目标程序独立于源程序（修改 → 再编译 → 运行）
  - 分析程序上下文，易于整体性优化
  - 性能更好（因此，核心代码通常C/C++）
- **解释**：源程序作为输入，边解释边执行
  - 不生成目标程序，可迁移性高
  - 逐句执行，很难进行优化
  - 性能通常不会太好



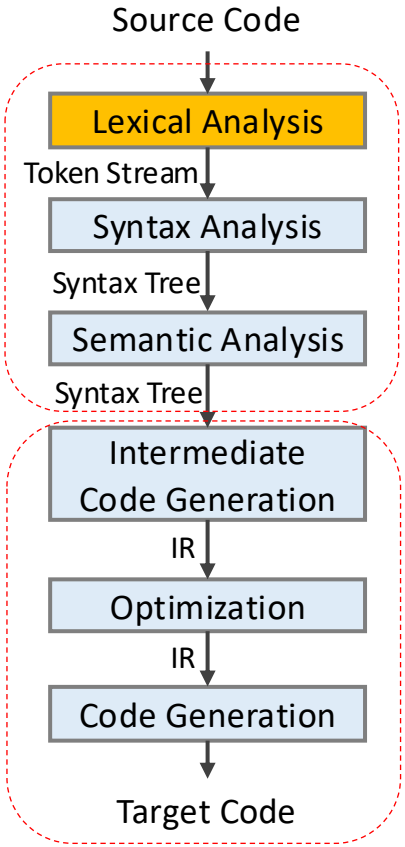


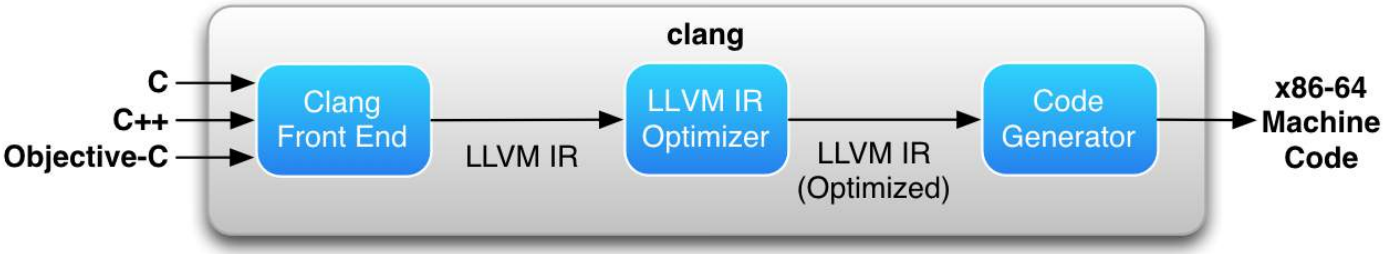
JIT[即时编译]

- **即时编译**（Just-In-Time Compiler）：运行时执行程序编译操作
  - 弥补解释执行的不足
    - 把翻译过的机器代码保存起来，以备下次使用
  - 传统编译（AOT, Ahead-Of-Time）：先编译后运行
- **JIT vs. AOT**
  - JIT具备解释器的灵活性
    - 只要有JIT编译器，代码即可运行
  - 性能上基本和AOT等同
    - 运行时编译操作带来一些性能上的损失
    - 但可以利用程序运行特征进行动态优化



Structure of a Typical Compiler[结构]





Example

```
void main(){
  int a, b, c;
  if (b == c)
    return 1;
}
```

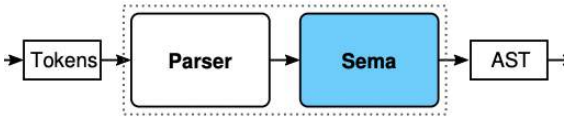
$clang -cc1 -dump-tokens test.c

```
void 'void'     [StartOfLine]  Loc=<parse.c:1:1>
identifier 'main'       [LeadingSpace] Loc=<parse.c:1:6>
l_paren '('             Loc=<parse.c:1:10>
r_paren ')'             Loc=<parse.c:1:11>
l_brace '{'             Loc=<parse.c:1:12>
int 'int'       [StartOfLine] [LeadingSpace]   Loc=<parse.c:2:3>
identifier 'a'  [LeadingSpace] Loc=<parse.c:2:7>
comma ','               Loc=<parse.c:2:8>
identifier 'b'  [LeadingSpace] Loc=<parse.c:2:10>
comma ','               Loc=<parse.c:2:11>
identifier 'c'  [LeadingSpace] Loc=<parse.c:2:13>
semi ';'                Loc=<parse.c:2:14>
if 'if'  [StartOfLine] [LeadingSpace]   Loc=<parse.c:3:3>
l_paren '('     [LeadingSpace] Loc=<parse.c:3:6>
identifier 'b'          Loc=<parse.c:3:7>
equalequal '=='  [LeadingSpace] Loc=<parse.c:3:9>
identifier 'c'  [LeadingSpace] Loc=<parse.c:3:12>
r_paren ')'             Loc=<parse.c:3:13>
return 'return'  [StartOfLine] [LeadingSpace]   Loc=<parse.c:4:5>
numeric_constant '1'    [LeadingSpace] Loc=<parse.c:4:12>
semi ';'                Loc=<parse.c:4:13>
r_brace '}'     [StartOfLine]  Loc=<parse.c:5:1>
eof ''          Loc=<parse.c:5:2>
```

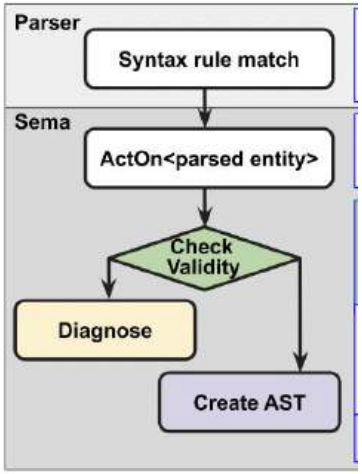
clang

$clang -Xclang -ast-dump -fsyntax-only test.c

```
`-FunctionDecl 0x27999470 <parse.c:1:1, line:5:1> line:1:6 main 'void ()'
  `-CompoundStmt 0x27999800 <col:12, line:5:1>
    |-DeclStmt 0x279996f8 <line:2:3, col:14>
    | |-VarDecl 0x27999570 <col:3, col:7> col:7 a 'int'
    | |-VarDecl 0x279995f0 <col:3, col:10> col:10 used b 'int'
    | `-VarDecl 0x27999670 <col:3, col:13> col:13 used c 'int'
    `-IfStmt 0x279997e8 <line:3:3, line:4:12>
      |-BinaryOperator 0x27999780 <line:3:7, col:12> 'int' '=='
      | |-ImplicitCastExpr 0x27999750 <col:7> 'int' <LValueToRValue>
      | | `-DeclRefExpr 0x27999710 <col:7> 'int' lvalue Var 0x279995f0 'b' 'int'
      | `-ImplicitCastExpr 0x27999768 <col:12> 'int' <LValueToRValue>
      |   `-DeclRefExpr 0x27999730 <col:12> 'int' lvalue Var 0x27999670 'c' 'int'
      `-ReturnStmt 0x279997d8 <line:4:5, col:12>
        `-ImplicitCastExpr 0x279997c0 <col:12> 'void' <ToVoid>
          `-IntegerLiteral 0x279997a0 <col:12> 'int' 1
```



Sema is tight coupling with parser



https://llvm.org/devmtg/2019-10/slides/ClangTutorial-Stulova-vanHaastregt.pdf

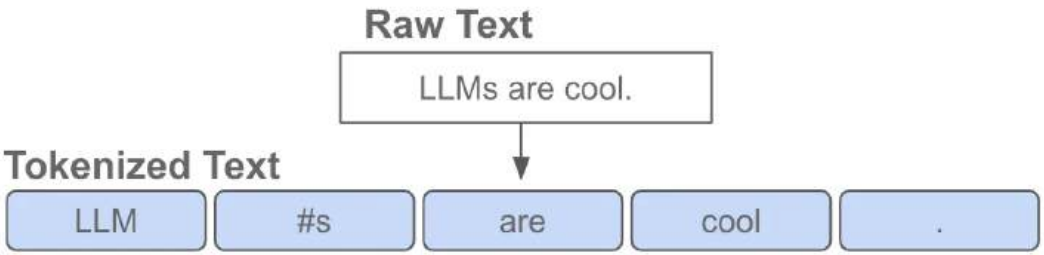
What is Lexical Analysis[词法分析]?

- Example:

```
/* simple example */
if (i == j)
  z = 0;
else
  z = 1;
```

- Input[输入]: a string of characters
  - "*if* (*i* == *j*)\n\t*z* = 0; \n*else*\n\t*z* = 1; \n"
- Goal[目标]: partition the string into a set of substrings
  - Those substrings are **tokens**
- Steps[步骤]
  - Remove comments: ~~/* simple example */~~
  - Identify substrings: 'if' '(' 'i' '==' 'j' ……
  - Identify **token classes**: (keyword, 'if'), (LPAR, '('), (id, 'i') ……

What is a token[词]?



https://www.53ai.com/news/LargeLanguageModel/2025022343062.html

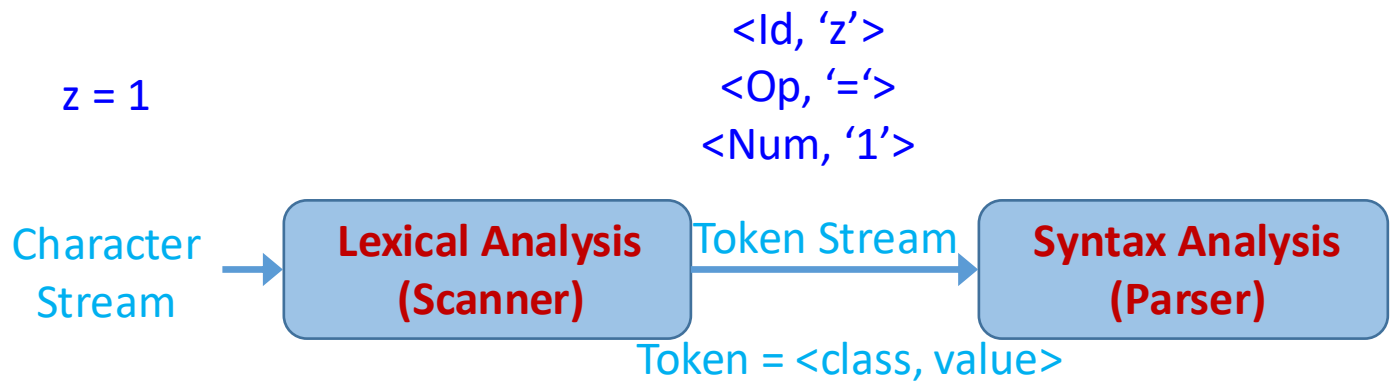
https://blog.csdn.net/zwqjoy/article/details/136813722

What is a token[词]?

- **Token**: a “word” in language (smallest unit with meaning)
  - Categorized into classes according to its role in language
  - Token classes in English[自然语言]
    - Noun, verb, adjective, …
  - Token classes in a programming language[编程语言]
    - Number, keyword, whitespace, identifier, …
- Each **token class** corresponds to a set of strings[类: 集合]
  - Number: a non-empty string of digits
  - Keyword: a fixed set of reserved words (“for”, “if”, “else”, …)
  - Whitespace: a non-empty sequence of blanks, tabs, newlines
  - Identifier: user-defined name of an entity to identify
    - Q: what are the rules in C language?

Lexical Analysis: Tokenization[分词]

- Lexical analysis is also called **Tokenization** (or, Scanner)[扫描器]
  - Partition input string into a sequence of tokens
  - Classify each token according to its role (token class)
    - **Lexeme**[词素]: an instance of the token class, e.g. 'z', '=', '1'
- Pass tokens to syntax analyzer (also called Parser)[分析器]
  - Parser relies on token classes to identify roles (e.g., a *keyword* is treated differently from an *identifier*)



Lexical Analyzer: Design[设计]

- Define a finite set of token classes[定义token类别]
  - Describe all items of interest
  - Depends on language, design of parser
  - "*if* (*i* == *j*)\n\t*z* = 0; \n*else*\n\t*z* = 1; \n"
    - Keyword, identifier, whitespace, integer

- Label which string belongs to which token class[识别]

```
if (i == j)
  z = 0;
else
  z = 1;
```

'==' or '='?

keyword or identifier?

Lexical Analyzer: Implementation[实现]

- An implementation must do two things
  - Recognize the token class the substring belongs to[识别分类]
  - Return the value or lexeme of the token[返回对应值]
- A token is a tuple (class, lexeme)[二元组]
- The lexer usually discards “non-interesting” tokens that don’t contribute to parsing[丢弃无意义词]
  - e.g., whitespace, comments
- If token classes are non-ambiguous, tokens can be recognized in a single left-to-right scan of input string
- Problem can occur when classes are ambiguous[歧义]

Ambiguous Tokens in C++

- C++ template syntax
  - Foo<Bar>
- C++ stream syntax
  - cin >> var

Template: a blueprint or formula for creating a generic class or a function.
Templates are expanded at compiler time, similar to macros.

```
Template <typename T>
T getMax(T x, T y) {
   return (x > y) ? x : y;
}

int main (int argc, char* argv[]) {
    getMax<int>(3, 7);
    getMax<double>(3.0, 2.0);
    getMax<char>('g', 'e');

    return 0;
}
```

- Ambiguity
  - Foo<Bar<Bar>>
  - cin >> var
  - Q: Is '>>' a stream operator or two consecutive brackets?

Look Ahead[展望]

- "look ahead" may be required to resolve ambiguity[展望消除歧义]
  - Extracting some tokens requires looking at the larger context or structure[需要上下文或语法结构]
  - Structure emerges only at parsing stage with parse tree[后一阶段才有]
  - Hence, sometimes feedback from parser needed for lexing
    - This complicates the design of lexical analysis
    - Should minimize the amount of look ahead

- Usually tokens do not overlap[通常无重叠]
  - Tokenizing can be done in one pass w/o parser feedback
  - Clean division between lexical and syntax analyses

Summary: Lexer

- Lexical analysis
  - Partition the input string to lexeme
  - Identify the token class of each lexeme
- Left-to-right scan => look ahead may be required
  - In reality, lookahead is always needed
  - The amount of lookahead should be minimized