EURO LLVM
Developers' Meeting
BERLIN 2025

Beyond Pattern-based Optimization:
What Can LLM Reshape *Auto-vectorization*?

Presenter: **Long Cheng**[1] (chenglong86@huawei.com)

Co-authors: Lu Li[2], Zhongchun Zheng[1, 5], Rodrigo Caetano de Oliveira Rocha[3], Wei Wei[1], Tianyi Liu[4], Li Zhou[1]

Affiliations: [1] Compiler Lab., Huawei Technologies Co., LTD, China
[2] Huawei Hongkong Research Institute, China HongKong
[3] Huawei Edinburgh Research Institute, UK Edinburgh
[4] Huawei Cambridge Research Institute, UK Cambridge
[5] Sun Yat-sen University, GuangZhou, China

HUAWEI

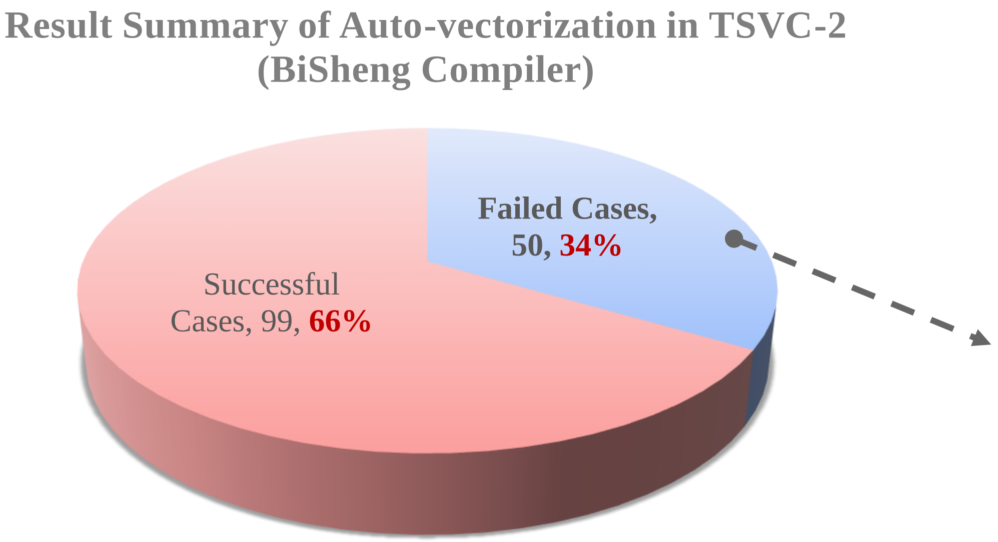
中山大學 SUN YAT-SEN UNIVERSITY

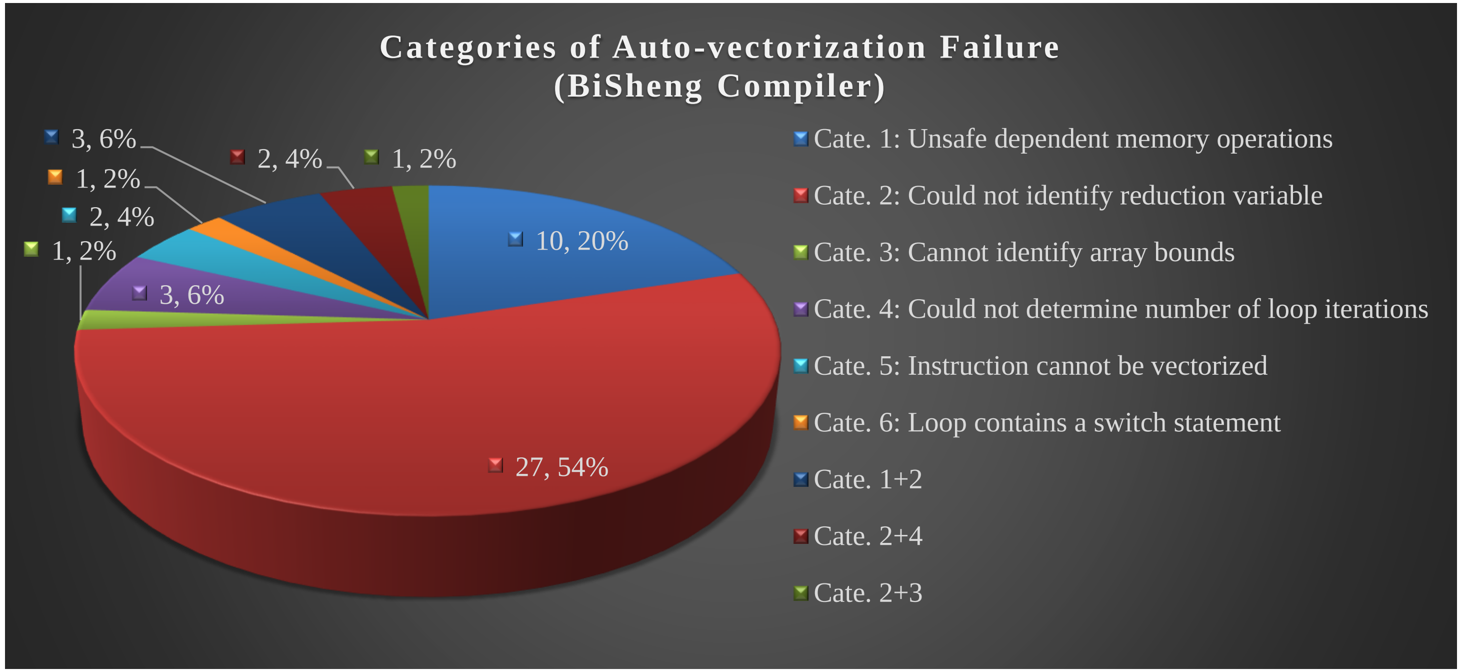
【More Information, please Refer to the Full Preprint Paper】

Zheng Z, Cheng L, Li L, et al. VecTrans: LLM Transformation Framework for Better Auto-vectorization on High-performance CPU[J]. arXiv preprint arXiv:2503.19449, 2025.*

Is It Good Enough for Auto-vectorization in Industrial Compilers?

- ➢ (1) Even for the simple benchmark - TSVC-2, there is still about **50/149** cases that cannot be vectorized by the LLVM-based industrial compiler(BiSheng Compiler) with **–O3** flag;
- ➢ (2) The dominant failure causes are: **limited analysis for memory dependencies and reduction**;
- ➢ (3) For **industrial applications** like HPC and mobile rendering, code scenarios are more complicated, which makes it hard to successfully **analyze the high-level semantics** purely utilizing traditional industrial compiler. **Hence, the answer to the title question is NO**.


Result Summary of Auto-vectorization in TSVC-2 (BiSheng Compiler)

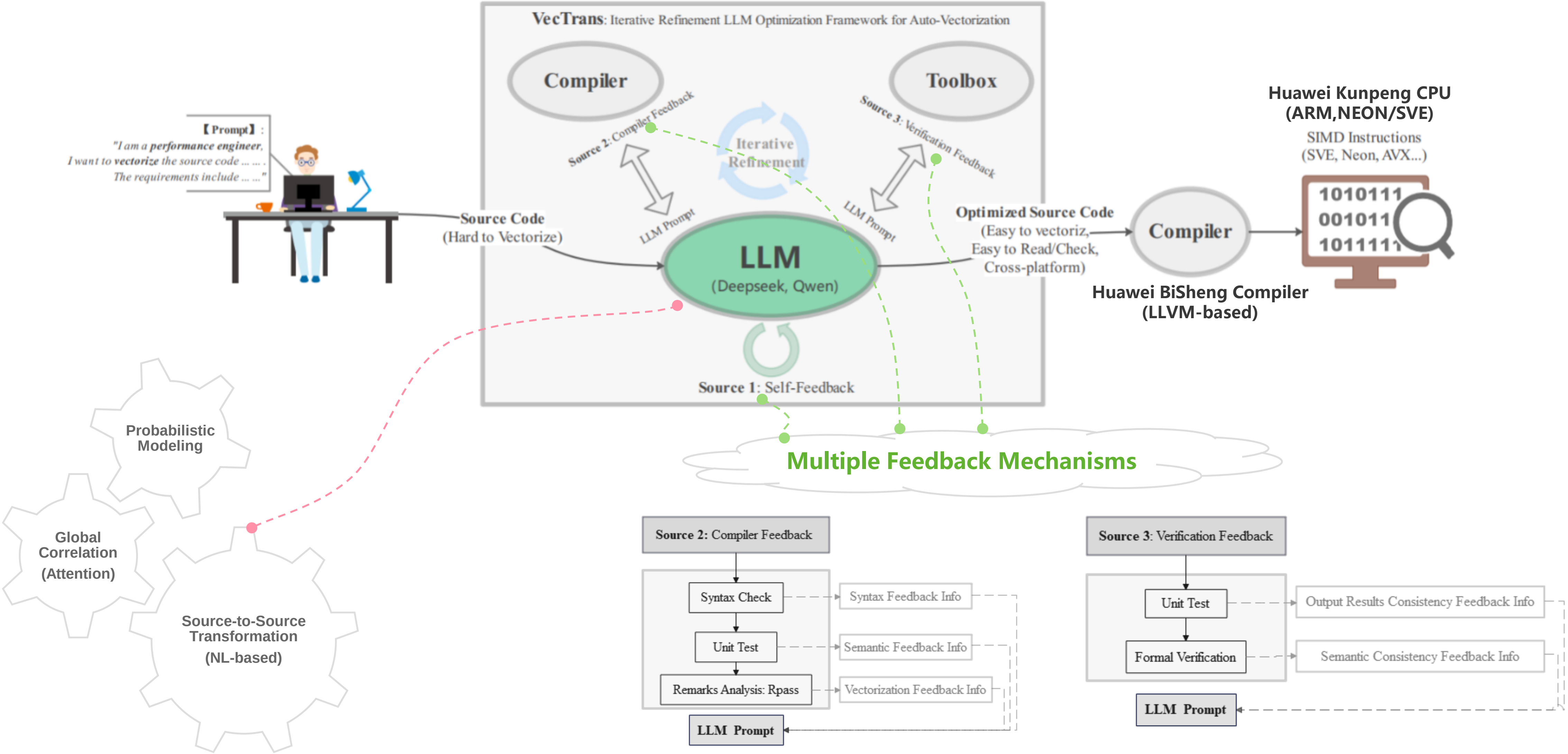

Categories of Auto-vectorization Failure (BiSheng Compiler)

- ➢ **Compiler**: BiSheng Compiler(LLVM-based, Industrial-level),
- ➢ **Hardware**: Kunpeng CPU-ARM(NEON/SVE)
- ➢ **Optimization Flags**: (1) -O3 -ffast-math; (2) -Rpass=loop-vectorize; (3) -Rpass-analysis=loop-vectorize

VecTrans: A LLM Compiler Agent Framework to Enhance Auto-vectorization

We built *VecTrans* to massively enhance the capability of auto-vectorization in traditional compilers. It has the following features:

(1) LLM-guided Source Code Transformations for Vectorization; *(2) Explores LLM-compiler Collaboration Paradigm;*
(3) Naively Support Cross-platform Verification; *(4) Generates Effective Results*



VecTrans: Iterative Refinement LLM Optimization Framework for Auto-Vectorization

【Prompt】:
*"I am a **performance engineer**,*
*I want to **vectorize** the source code ... .*
The requirements include ... ..."

Source Code
(Hard to Vectorize)

Compiler

Toolbox

Source 2: Compiler Feedback

Source 3: Verification Feedback

Iterative Refinement

LLM Prompt

LLM Prompt

LLM
(Deepseek, Qwen)

Source 1: Self-Feedback

Optimized Source Code
(Easy to vectoriz,
Easy to Read/Check,
Cross-platform)

Compiler

Huawei BiSheng Compiler
(LLVM-based)

Huawei Kunpeng CPU
(ARM,NEON/SVE)

SIMD Instructions
(SVE, Neon, AVX...)

1010111
001011
1011111

Probabilistic Modeling

Global Correlation (Attention)

Source-to-Source Transformation (NL-based)

Multiple Feedback Mechanisms

Source 2: Compiler Feedback

Syntax Check → Syntax Feedback Info

Unit Test → Semantic Feedback Info

Remarks Analysis: Rpass → Vectorization Feedback Info

LLM Prompt

Source 3: Verification Feedback

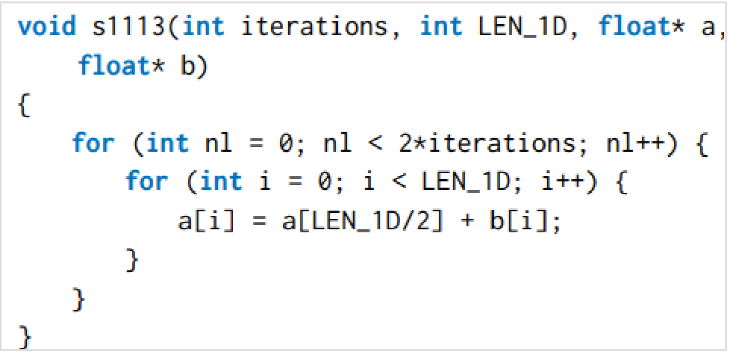
Unit Test → Output Results Consistency Feedback Info

Formal Verification → Semantic Consistency Feedback Info

LLM Prompt

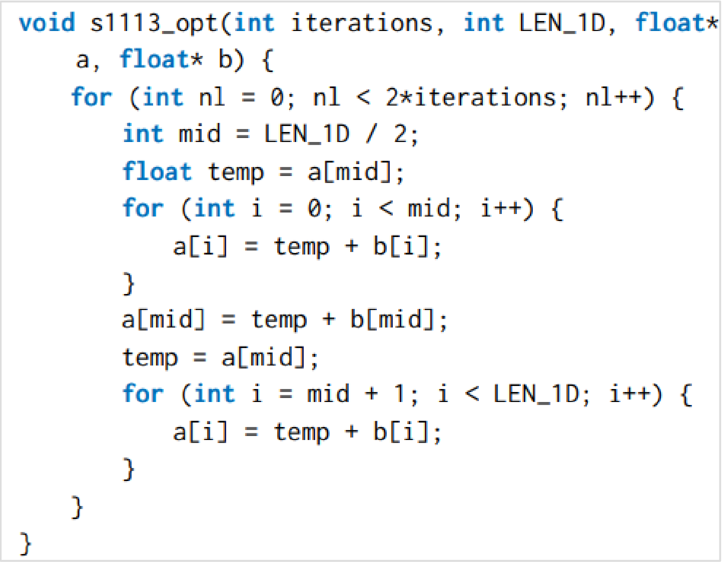
Current Results

Benchmark Code(TSVC-2, s1113)



```
void s1113(int iterations, int LEN_1D, float* a,
    float* b)
{
    for (int nl = 0; nl < 2*iterations; nl++) {
        for (int i = 0; i < LEN_1D; i++) {
            a[i] = a[LEN_1D/2] + b[i];
        }
    }
}
```

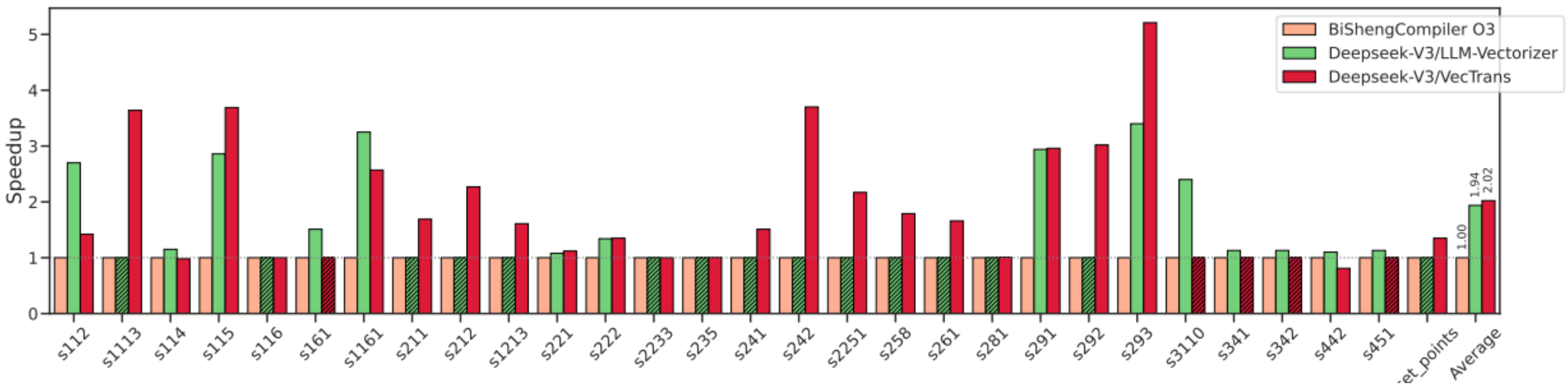
VecTrans



```
void s1113_opt(int iterations, int LEN_1D, float*
    a, float* b) {
    for (int nl = 0; nl < 2*iterations; nl++) {
        int mid = LEN_1D / 2;
        float temp = a[mid];
        for (int i = 0; i < mid; i++) {
            a[i] = temp + b[i];
        }
        a[mid] = temp + b[mid];
        temp = a[mid];
        for (int i = mid + 1; i < LEN_1D; i++) {
            a[i] = temp + b[i];
        }
    }
}
```

Comparison with SOTA(TSVC-2)

LLM-Vectorizer(C -> SIMD Intrinsic)
Taneja J, Laird A, Yan C, et al. *Llm-vectorizer: Llm-based verified loop vectorizer.* CGO 2025: 137-149.



Auto-vectorization Ablation Experiments

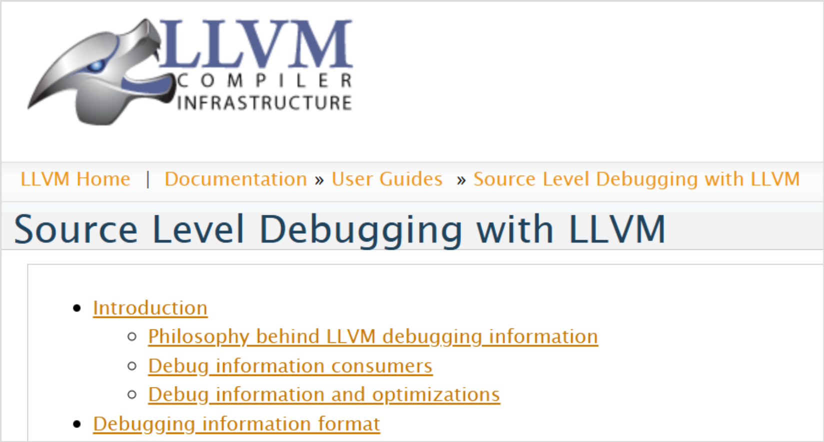
(BiSheng Compiler+Kunpeng CPU-Neon/SVE)

Configurations	Success Ratio	Average Iteration Times
DeepSeek-V3/**VecTrans**	**46.2%**	8.76
DeepSeek-V3/LLM-Vectorizer	28.8%	13.39
DeepSeek-V3/Base Model	17.3%	5.41
Qwen2.5-32B/VecTrans	34.6%	13.94
Qwen2.5-72B/VecTrans	38.5%	12.26
DeepSeek-V3/Without Formal Verification	32.7%	8.49
DeepSeek-V3/**Without Compiler Feedback**	**21.2%**	5.21
DeepSeek-V3/Without Unit Test	25.0	9.66

Discussion and Future Work

1. If we can open the debug information in LLVM infrastructure to designate more precise program analysis information to LLM, we believe that the concept of ***LLM as a Compiler optimizer*** will become more practical for industrial applications;
2. The current VecTrans work will be open source in openEuler community. (https://gitee.com/openeuler/llvm-project)

OpenEuler https://www.openeuler.org/en/



LLVM COMPILER INFRASTRUCTURE

LLVM Home | Documentation » User Guides » Source Level Debugging with LLVM

Source Level Debugging with LLVM

- Introduction
  - Philosophy behind LLVM debugging information
  - Debug information consumers
  - Debug information and optimizations
- Debugging information format

100% Support for Mainstream Computing Architectures
Arm, x86, RISC-V, SW-64, LoongArch
NPUs, GPUs, DPUs
100+ devices, 300+ boards
Server
Cloud & Cloud Native
Edge Computing
Embedded

openEuler is an open source project incubated and operated by the OpenAtom Foundation.