中北大学学报(自然科学版)
JOURNAL OF NORTH UNIVERSITY OF CHINA(NATURAL SCIENCE EDITION)
2026年 第47卷 第5期(总第229期)
Vol. 47 No. 5 2026(Sum No. 229)

文章编号：1673-3193(2026)05-0729-11 J. North Univ. China, Nat. Sci. Ed., 2026, 47(5): 729-739.

面向OpenMP自动并行化的结构化约束驱动制导语句补全研究

申园园，李柏汛，尚 磊，曾建潮，秦品乐*
（中北大学 计算机科学与技术学院，山西 太原 030051）

摘 要：针对串行程序自动并行化中并行语义识别不充分与OpenMP制导语句自由生成可控性不足的问题，本文提出一种结构化约束驱动的两阶段OpenMP自动并行化方法。传统静态分析方法在复杂数据依赖、非规范循环和边界样本中容易表现出过度保守或误判，而通用大模型自由生成pragma时缺乏显式的子句边界和变量约束，容易产生误并行、子句误用和变量幻觉。为此，本文构建并行属性预测模型CurParCL-X，在代码文本与数据流图联合表征基础上，引入层次化属性结构、任务难度驱动的动态课程学习、三视图一致性对比学习和错误驱动反馈机制，获得面向下游补全的属性级先验。随后根据并行化决策、子句状态、属性置信度和变量候选集合构造结构化约束信号，并进一步构造约束派生候选指令，将其与候选循环共同组织为候选指令引导的结构化prompt，使大模型在显式语义边界内完成制导语句补全。实验结果表明，CurParCL-X在并行属性预测任务中优于现有基线。结构化约束信号能够有效降低无约束生成中的误生成问题，使通用大模型在不进行额外微调的条件下获得更稳定、可解释的并行制导语句补全能力。

关键词：自动并行化；并行性检测；共享内存并行性；OpenMP；大语言模型；对比学习
中图分类号：TP391.41 **文献标识码：**A **doi:** 10.62756/jnuc.issn.1673-3193.2026.05.0016
引用格式：申园园，李柏汛，尚磊，等. 面向OpenMP自动并行化的结构化约束驱动制导语句补全研究[J]. 中北大学学报(自然科学版)，2026，47(5)：729-739.
Shen Yuanyuan, Li Boxun, Shang Lei, et al. Study on structured constraint-driven guided statement completion for OpenMP automatic parallelization[J]. Journal of North University of China(Natural Science Edition), 2026, 47(5): 729-739.

Study on Structured Constraint-Driven Guided Statement Completion for OpenMP Automatic Parallelization

Shen Yuanyuan, Li Boxun, Shang Lei, Zeng Jianchao, Qin Pinle*
(School of Computer Science and Technology, North University of China, Taiyuan 030051, China)

Abstract: To address the issues of insufficient parallel semantic recognition and lack of controllability in the free generation of OpenMP directives during the automatic parallelization of serial programs, this paper proposes a two-phase OpenMP automatic parallelization method driven by structural constraints. Traditional static analysis methods tend to be overly conservative or produce false positives when complex data dependencies, non-standard loops, and boundary cases are processed. While general-purpose large models lack explicit clause

收稿日期：2026-05-20
基金项目：山西省基础研究计划项目(202403021212168)
作者简介：申园园(1994—)，女，讲师，博士，主要从事并行计算、程序语言处理方面的研究。E-mail：shenyuanyuan@nuc.edu.cn。
* **通信作者：**秦品乐(1978—)，男，教授，博士，主要从事机器视觉、大数据的研究。E-mail：qpl@nuc.edu.cn。

boundaries and variable constraints when freely generating pragmas, leading to false parallelization, clause misuse, and variable phantom effects. To address this, we construct a parallel attribute prediction model, CurParCL-X. Based on a joint representation of code text and data flow graphs, we introduce a hierarchical attribute structure, task-difficulty-driven dynamic curriculum learning, three-view consistency comparison learning, and an error-driven feedback mechanism to obtain attribute-level priors for downstream completion. Subsequently, structured constraint signals are constructed based on parallelization decisions, clause states, attribute confidence, and variable candidate sets. Constraint-derived candidate instructions are further generated and organized together with candidate loops into candidate-instruction-guided structured prompts, enabling the large model to complete guided statement completion within explicit semantic boundaries. Experimental results demonstrate that CurParCL-X outperforms existing baselines in parallel attribute prediction tasks. The structured constraint signal effectively reduces misgenerations in unconstrained generation, enabling general-purpose large models to achieve more stable and interpretable parallel guidance statement completion capabilities without additional fine-tuning.

Key words: automatic parallelization; parallelism detection; shared-memory parallelism; OpenMP; large language model; contrastive learning

0 引 言

随着多核与异构体系结构的广泛部署，如何将大量串行程序自动转换为正确且高效的并行程序，已成为高性能计算中的关键问题。OpenMP具有接口清晰、使用灵活的特点，是共享内存并行编程中最常用的方法之一，但从串行代码中自动识别可并行区域并进一步生成正确的OpenMP制导语句，仍然具有较高难度。对于自动并行化而言，系统不仅需要判断循环是否具备并行执行潜力，还需要进一步识别变量私有化、归约以及相关共享访问模式。若前置并行性预测结果不准确，则后续pragma生成、编译验证与性能优化都会受到直接影响。因此，并行性检测既是自动并行化的入口任务，也是决定下游生成质量与工具可信性的基础环节[1]。

传统的自动并行化方法主要依赖静态分析和动态分析。多面体框架和源到源编译器能够基于循环边界、访存关系和数据依赖等信息，对具有仿射内存访问的规则循环进行形式化分析，并自动插入OpenMP制导语句；动态性能分析工具则通过运行时依赖检测获取程序的实际执行行为，在一定程度上缓解了静态分析过于保守的问题[2-3]。这类方法具有分析规则明确、结果可解释性较强且便于与编译器优化流程集成等优点，尤其适用于循环结构规则、数组下标可解析且数据依赖关系清晰的程序。在满足分析假设的情况下，传统方法还能够利用显式依赖规则降低错误并行化风险。

传统方法的分析能力通常受限于预设规则、程序中间表示和依赖分析精度。静态分析在函数调用、复杂控制流和非规范循环时，往往难以获得完整且精确的依赖信息，因此容易采取保守策略并遗漏潜在并行机会。动态分析虽然能够利用运行时信息减少部分保守判断，但其分析结果依赖具体输入和执行路径，同时会引入额外的运行时检测开销，难以保证对未覆盖路径的完整分析。因此，传统方法在安全性和可解释性方面具有优势，但对复杂程序语义的适应能力仍然有限。

近年来，数据驱动的程序分析为自动并行化带来了新的研究契机。与传统方法相比，数据驱动方法能够从训练样本中学习代码结构、变量关系及潜在并行模式，从而降低对人工特征和启发式规则的依赖。基于图的程序表示方法及图神经网络(GNN)能够更有效地建模代码结构及其语义关系[4]。但是，基于GNN的方法通常依赖图的构建质量，当程序图遗漏别名关系、跨过程依赖或隐式内存访问时，学习到的表示仍可能不完整。对比学习通过对齐语义相关的不同视图来增强表示鲁棒性[5-6]，而课程学习则通过按照由易到难的顺序组织训练样本来提升优化过程的稳定性[7]。OMPar进一步将并行性检测与pragma生成串联起来，形成模块化自动并行化流水线[8]。AutoParLLM则利用GNN生成与输入程序相关的上下文，指导大语言模型进行零样本并行代码的生成[9]。

基于上述分析，本文提出一种结构化约束驱动

的两阶段 OpenMP 自动并行化框架，以兼顾数据驱动方法的复杂语义识别能力与显式约束方法的可控性。第一阶段在 CurParCL 基础上，从属性结构一致性、难样本适应和误差感知反馈三方面进行增强，构建并行属性预测模型 CurParCL-X。其目标不仅要提升并行属性分类的性能，还要输出可用于下游补全的结构化约束信号。第二阶段将上述属性先验与数据流分析得到的候选变量集合相结合，构造结构化约束信号和约束派生候选指令，使大语言模型不再进行完全开放的制导语句生成，而是在显式语义边界和变量范围内完成受控补全。

本文的主要贡献如下：

1）在 CurParCL 框架基础上提出了一种增强型并行属性预测模型 CurParCL-X，通过逻辑约束的层次化属性结构、难度感知动态课程采样、多视图一致性对比学习和误差感知反馈机制，提升 3 类并行属性预测的准确性、稳定性与结构一致性，为后续 OpenMP 制导语句受控补全提供属性级约束先验。

2）提出一种面向 OpenMP 制导语句受控补全的结构化约束信号构建方法。将 CurParCL-X 输出的并行化决策、子句状态和置信度信息，与数据流图分析得到的候选变量集合相结合，形成结构化中间表示，从而提升约束信号的可解释性和约束能力。

3）提出一种候选指令引导的 OpenMP 制导语句受控补全策略。对于不具备并行化条件的候选循环，直接跳过大模型调用，避免无效推理和负样本生成。对于可并行循环，框架将结构化约束信号与候选循环代码构造为候选引导的结构化 prompt，对于不满足决策或格式约束的，则通过修复式提示（repair prompt）和直接候选提示（direct candidate prompt）进行修复，从而提升结果的可控性和稳定性。

1　自动并行化

1.1　问题定义与整体框架

本文面向自动并行化中的 OpenMP 制导语句补全任务，提出一种基于结构化约束信号的两阶段轻量化方法，具体框架如图 1 所示。与直接对大语言模型进行领域微调[10-13]，或完全依赖大语言模型自由生成 OpenMP 制导语句的方法不同[14-15]，本文将自动并行化过程拆分为“并行语义识别”和“约束驱动补全”两个阶段。第一阶段利用 CurParCL-X，从候选循环的代码文本和数据流图中识别 3 类关键并行属性。第二阶段将结构化约束信号转化为候选指令引导的结构化 prompt，进而驱动大模型完成 OpenMP 制导语句补全。通过这一设计，大语言模型不再承担完整的并行语义推断任务，而是聚焦于给定约束下的候选指令补全，从而降低自由生成中的子句歧义、变量绑定错误和负样本误生成风险。

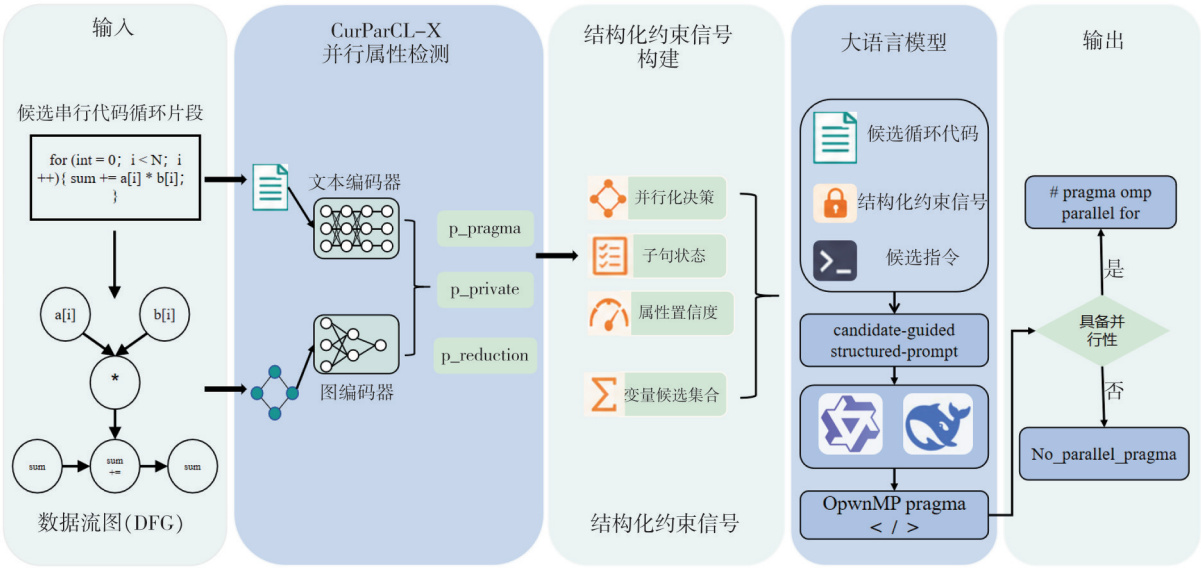


图 1　结构化约束驱动的两阶段 OpenMP 自动并行化框架图

Fig. 1　Diagram of a two-phase OpenMP automatic parallelization framework driven by structural constraints

1.2 CurParCL-X

1.2.1 逻辑约束层次化结构

现有并行属性预测方法通常将pragma、private和reduction作为3个独立标签同时输出，这种做法虽然实现简单，但容易忽略它们在OpenMP语义上的依赖关系。例如，当某个循环本身不适合插入并行指令时，其private或reduction属性通常也不应被使用。为此，本文将原来的并列多标签输出改为层次化结构。双分支编码后的表示为z_i，则3个标签的预测过程定义为

$$p_i^{\mathrm{p}}=\sigma(W_{\mathrm{p}}z_i+b_{\mathrm{p}}), \tag{1}$$

$$p_i^{\mathrm{v}}=\sigma(W_{\mathrm{v}}[z_i;\ p_i^{\mathrm{p}}]+b_{\mathrm{v}}), \tag{2}$$

$$p_i^{\mathrm{r}}=\sigma(W_{\mathrm{r}}[z_i;\ p_i^{\mathrm{p}};\ p_i^{\mathrm{v}}]+b_{\mathrm{r}}), \tag{3}$$

式中：p_i^{p}，p_i^{v}和p_i^{r}分别表示样本的pragma、private和reduction预测概率。该设计使后续属性能够感知前序属性状态，从而降低标签之间的语义冲突。为进一步约束组合，加入逻辑一致性正则项

$$L_{\mathrm{logic}}=\frac{1}{B}\sum_{i=1}^{B}(p_i^{\mathrm{v}}(1-p_i^{\mathrm{p}})+p_i^{\mathrm{r}}(1-p_i^{\mathrm{p}})), \tag{4}$$

式中：B为批次大小。用于抑制无pragma时却高置信预测private和reduction的不合理输出，通过层次化结构和逻辑约束，模型输出不再只是数值上独立的3个概率，而是更加符合并行语义的结构化属性预测结果。

1.2.2 任务难度驱动的动态课程学习

并行属性预测任务中的样本难度并不只由代码长度决定，较短循环也可能包含复杂的数据依赖。因此，仅依据代码长度组织课程学习难以充分反映并行属性预测的真实困难来源。为此本文构建一种任务难度驱动的动态课程学习策略，从代码规模、数据流复杂度和历史预测误差三个方面综合评估样本难度，并据此动态调整训练样本的采样顺序。将样本难度定义为多因素组合

$$D(x_i)=\alpha D_{\mathrm{len}}(x_i)+\beta D_{\mathrm{dfg}}(x_i)+\gamma D_{\mathrm{unc}}(x_i)+\delta D_{\mathrm{hist}}(x_i), \tag{5}$$

式中：D_{len}表示代码长度；D_{dfg}表示数据流图复杂度；D_{hist}表示样本的历史误差强度。对于候选循环样本x_i，首先计算代码长度复杂度，用于衡量代码token数量。随后根据数据流图中的节点数和依赖边数计算DFG结构复杂度D_{dfg}，用于刻画变量依赖关系的复杂程度。与此同时，为了反映模型在训练过程中的动态判别难度，本文记录模型在历史轮次中对样本的预测误差，并将其表示为历史误差强度D_{hist}。为消除不同指标差异，本文分别对代码长度复杂度、DFG结构复杂度和历史误差强度进行归一化处理，并将三者加权融合为综合难度分数

$$d_i^{(t)}=\lambda_1 D_{\mathrm{len}}+\lambda_2 D_{\mathrm{dfg}\,i}+\lambda_3 D_{\mathrm{hist}}, \tag{6}$$

式中：λ_1、λ_2和λ_3为3类因素的权重。基于综合难度分数$d_i^{(t)}$，本文将训练样本动态划分为easy、medium和hard三类，在训练过程中，课程采样策略随训练轮次动态调整。训练初期优先采样easy样本，使其先学习结构较简单、数据依赖较清晰的循环模式；随着训练推进，medium和hard样本的采样比例逐步提高，使模型逐渐关注DFG结构复杂、属性边界模糊或历史误判频繁的高风险样本。通过这种方式，模型能够在保持训练稳定性的同时增强对复杂依赖模式和边界样本的适应能力，从而提升并行属性预测结果的稳定性和结构一致性。

1.2.3 三视图一致性对比学习

为增强并行属性预测模型对代码表层变化和数据流结构扰动的鲁棒性，本文引入多视图一致性对比学习机制，将easy、medium和hard三类训练样本统一表示为原始文本视图、增强文本视图和增强图视图，使模型同时对表层代码变化和轻度结构扰动保持稳定判别能力。将整体对比学习目标统一写为

$$L_{\mathrm{con}}=\lambda_{\mathrm{tt}}L_{\mathrm{tt}}+\lambda_{\mathrm{tg}}L_{\mathrm{tg}}+\lambda_{\mathrm{gg}}L_{\mathrm{gg}}, \tag{7}$$

式中：L_{tt}为原始文本与增强文本一致性损失；L_{tg}为文本与图一致性损失；L_{gg}为原始图与增强图一致性损失。这一设计的关键作用在于使模型学习到同一代码在不同扰动下应保持一致，代码序列语义与结构依赖表示应保持一致，而且图结构在轻度扰动下仍应保持稳定。因此，模型在面对异构编码风格、复杂依赖交互和结构扰动时，能够获得更稳健的表示空间。

1.2.4 反馈增强机制

现有并行性检测方法在测试阶段往往会产生一批假阴性样本，这些样本通常集中反映模型在边界区域、少数类组合和复杂依赖模式上的薄弱环节。然而，大多数方法未将其进一步转化为训练信号。为此，本文提出错误驱动反馈增强机制。对于测试阶段产生的假阴性样本构建一个集合，在后续的训练中对该集合的样本赋予更高的采样权重。该机制的作用是将模型曾经预测失败的样

本重新纳入重点学习对象，使训练过程能够持续针对高风险模式进行修正。结合前述难度驱动课程学习，假阴性样本成为后续训练中的重要难样本来源，从而形成训练-测试-训练的优化。综上所述，整体任务优化目标为

$$L = L_{\text{cls}} + \lambda_{\text{con}} L_{\text{con}} + \lambda_{\text{logic}} L_{\text{logic}}。\tag{8}$$

将图-文本双分支并行性检测框架进一步优化为基于结构约束和难度感知反馈的增强型模型，为之后的OpenMP制导语句生成提供更高质量的约束信号。

综上，本文CurParCL-X实现框架如图 2 所示。

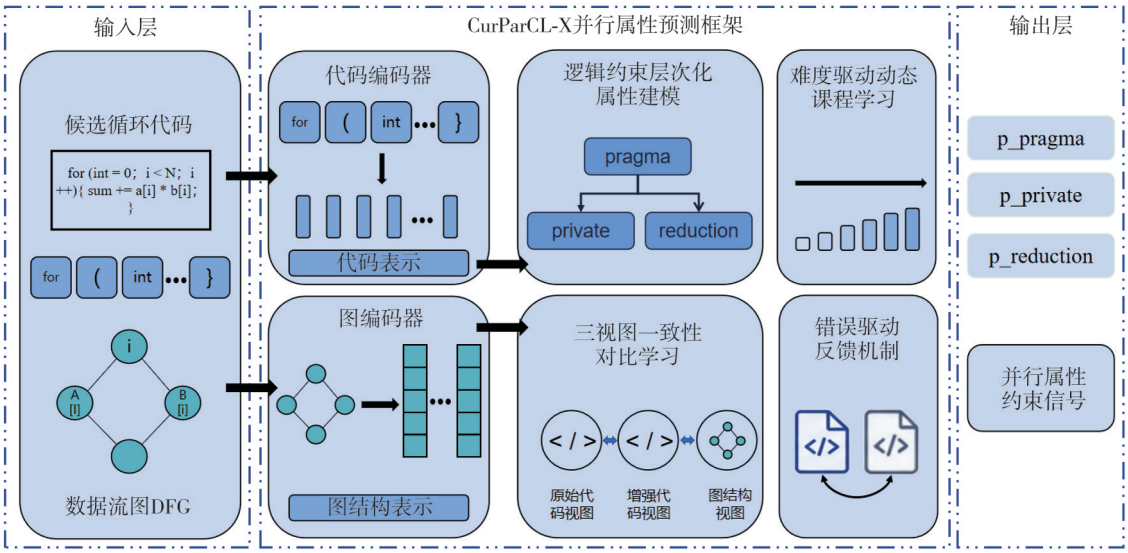


图 2　CurParCL-X并行属性预测框架

Fig. 2　Prediction framework of CurParCL-X parallel attribute

1.3　约束信号构建

在本文框架中，约束信号是连接并行属性预测与OpenMP制导语句补全之间的结构化中间表示。OpenMP制导语句生成需要同时解决4类决策问题：1）当前串行代码是否具备并行性；2）是否需要生成private或reduction子句；3）子句应作用于哪些变量；4）模型对上述判断的置信程度如何。若完全依赖通用大模型自由生成，则这些决策均需要由大模型从代码上下文中隐式推断，容易产生不必要的错误子句、错误变量绑定或错误归约操作符。为此，本文将CurParCL-X的属性预测结果与DFG变量依赖分析相结合，构建面向生成任务的结构化约束信号。对于候选循环，本文定义结构化约束信号为

$$S(x)=\{D_\theta(x), A_\theta(x), P_\theta(x), V_{\text{dfg}}(x), R(x)\},\tag{9}$$

式中：$D_\theta(x)$为并行化约束决策；$A_\theta(x)$为子句类型约束；$P_\theta(x)$为置信度约束，用于刻画并行属性预测结果的可靠程度，作为软约束参与prompt的构建，为边界样本提供额外判断依据；$V_{\text{dfg}}(x)$为变量级候选约束；$R(x)$为补全阶段的逻辑规则，用于限制大模型输出的范围，将开放式生成空间压缩为受控的制导语句补全空间。

1.3.1　并行化决策约束

为了避免大模型在不可并行或不应该并行的串行代码上生成错误的制导语句，减少大模型的推理次数，降低生成延迟，本文对第一阶段pragma属性的概率设定阈值判断得到生成决策。当概率低于阈值时，表示为$D_\theta(x)=Skip$，直接终止当前串行代码的后续生成过程。

1.3.2　子句类型约束

当$D_\theta(x)=Generate$时，进一步构建子句类型约束，定义为

$$A_\theta(x)=\{Required, Optional, Forbidden\},\tag{10}$$

式中：*Required*表示在候选变量可用且不违反代码语义时应包含该子句；*Forbidden*表示不可以引入该子句；*Optional*表示弱约束状态，对该子句不进行硬性要求。相较于二值标签，这样的约束可以避免边界样本陷入刚性规则，使得约束信号构建具备容错性。

1.3.3　变量级候选约束

无论是private还是reduction子句都需要明确的变量参数，为保持数据依赖的一致性，构建集合如下

$$V_{dfg}(x)=\{V_{private}(x), V_{reduction}(x), O_{reduction}(x)\}, \tag{11}$$

式中：$V_{private}(x)$表示private候选变量集合；$V_{reduction}(x)$表示reduction候选变量集合；$O_{reduction}(x)$表示候选规约操作符集合。候选变量集合不是最终变量集合，其作用是为大模型限定变量搜索范围，减少变量幻觉和错误变量绑定。

完整的结构化约束信号以结构化记录形式保存，主要包括decision、attributes、variables和logic_rules四类字段。其中，decision用于描述当前样本是否进入补全流程；attributes给出三类属性预测标签、子句状态和置信度；variables给出候选变量集合；logic_rules描述补全阶段必须满足的约束规则，例如禁止生成Forbidden子句、禁止引入代码中不存在的变量等。该结构化记录为后续候选指令构造和prompt生成提供统一输入，其具体形式如下：

```
{
"decision": {
"status": "Generate",
"p_pragma": 0.9988
},
"attributes": {
"private": {
"status": "Forbidden",
"confidence": 0.021
},
"reduction": {
"status": "Required",
"confidence": 0.964
}
},
"variables": {
"private_candidates": [],
"reduction_candidates": [
{
"variable": "sum",
"operator": "+",
"evidence": "sum += ..."
}
],
"shared_candidates": ["n", "step"]
},
"logic_rules": {
"forbidden_clause_must_not_be_generated": true,
"do_not_introduce_nonexistent_variables": true
}
}
```

1.4 基于候选指令引导的OpenMP制导语句补全

在第二阶段，根据候选循环代码和结构化约束信号构造为约束派生候选指令，其基础形式是#pragma omp parallel for。再将三者共同组织为结构化约束prompt。该设计将OpenMP制导语句生成问题转化为显式语义约束下的制导语句补全问题。结构化约束prompt基本形式如下：

```
Decision: {status}
Pragma confidence: {p_pragma}
Private status: {private_status}
Private candidates: {compact_list(private_candidates)}
Reduction status: {reduction_status}
Reduction candidates: {', '.join(red_parts) if red_parts else 'none'}
Shared candidates: {compact_list(shared_candidates)}
Constraint-derived candidate directive: {candidate}
Rules
1. {decision_rule}
2. Do not generate clauses marked Forbidden.
3. Prefer the candidate directive unless it violates the code or constraints.
4. Do not invent variables.
5. If a clause is Required, include it when the candidate provides the needed variable.
C loop:
{code.strip()}
Final pragma:
```

为提升输出的格式一致性和鲁棒性，进一步采用多阶段修复策略。对于Generate样本，框架首先使用候选指令引导的prompt进行补全；若模型初次输出不满足单行格式、决策约束或OpenMP制导语句格式，则进入修复阶段，构造repair prompt将上一轮无效输出作为错误上下文，并要求模型根据当前决策状态和候选指令重写为合法的结果，引导模型修正输出；若仍然无效，则使用direct candidate prompt要求模型直接输出约束派生候选指令。通过上述机制，第二阶段大语言模型的任务由开放式制导语句生成转化为结构化约束下的候选指令受控补全，使整个自动并行化流程具有更强的可控性和可解释性。repair prompt构造如下：

```
Rewrite the final answer as exactly ONE valid line.
{extra}
Use this candidate unless clearly unsafe:
{candidate}
Allowed output for this sample:
```

{allowed}
Final answer only：

2　实　验

2.1　实验设置

所有实验均在 NVIDIA A800 GPU 上完成，第一阶段使用 Open-OMP-Plus 数据集，并按 8∶1∶1的比例划分为训练集、验证集与测试集。对比学习相关超参数设置为：动量系数 m=0.95，温度参数 T=0.25，队列大小 K=2048。为避免数据重复，剔除了与训练集存在重叠的基准样本。第一阶段实验在统一的数据划分和训练设置下对并行属性预测模型进行评估。第二阶段使用经过处理的 OpenMP 专用测试集来评测模型生成能力，以 parallel for、private、reduction 等并行指令为核心，契合任务目标。

2.2　评价指标

为统一评价第一阶段并行属性预测和第二阶段 OpenMP 制导语句补全的性能，本文采用准确率（Accuracy）R_{Acc}、精确率（Precision）R_{Pre}、召回率（Recall）R_{Rec} 和 $F1$ 值（$F1$-score）作为主要评价指标。对于任一二分类任务，N_{TP}（True Positive，TP）表示正类样本被正确预测为正类的数量，N_{TN}（True Negative，TN）表示负类样本被正确预测为负类的数量，N_{FP}（False Positive，FP）表示负类样本被错误预测为正类的数量，N_{FN}（False Negative，FN）表示正类样本被错误预测为负类的数量。各评价指标定义如下：

$$R_{\mathrm{Acc}}=\frac{N_{\mathrm{TP}}+N_{\mathrm{TN}}}{N_{\mathrm{TP}}+N_{\mathrm{TN}}+N_{\mathrm{FP}}+N_{\mathrm{FN}}},\tag{12}$$

$$R_{\mathrm{Pre}}=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FP}}},\tag{13}$$

$$R_{\mathrm{Rec}}=\frac{N_{\mathrm{TP}}}{N_{\mathrm{TP}}+N_{\mathrm{FN}}},\tag{14}$$

$$F1=\frac{2\times R_{\mathrm{Prec}}\times R_{\mathrm{Rec}}}{R_{\mathrm{Pre}}+R_{\mathrm{Rec}}},\tag{15}$$

式中：R_{Acc} 用于衡量全部样本中预测正确的比例；R_{Pre} 反映模型预测为正类的样本中实际属于正类的比例，可用于衡量误报情况；R_{Rec} 反映实际正类样本被模型正确识别的比例，可用于衡量漏报情况；$F1$ 值是 R_{Pre} 与 R_{Rec} 的调和平均，用于综合评价模型在误报和漏报之间的平衡。

在第一阶段并行属性预测实验中，pragma、private 和 reduction 分别作为独立的二分类任务进行评价。对于 pragma 属性，正类表示候选循环可以生成 OpenMP 并行制导语句；对于 private 和 reduction 属性，正类分别表示对应循环需要 private 子句或 reduction 子句。

在第二阶段 OpenMP 制导语句补全实验中，将需要生成 OpenMP 制导语句的样本视为正类，将不需要生成制导语句的样本视为负类。N_{TP} 表示应生成且模型正确生成制导语句的样本数；N_{FP} 表示原本不应生成但模型生成了制导语句的样本数；N_{TN} 表示原本不应生成且模型未生成制导语句的样本数；N_{FN} 表示应生成但模型未能生成制导语句的样本数。

2.3　并行性预测实验结果

为全面评估代码模型性能，将其与多种代表性基线模型进行对比，包括面向特定任务的模型与预训练代码模型。表 1 汇总了各个模型的性能结果，其中，本文提出的方法在该任务上取得了最优整体性能。

表 1　不同模型在并行性检测中的性能对比

Tab. 1　Performance comparison of different models in parallelism detection

模型	评测指标								
	R_{Acc}/%			R_{Pre}/%			R_{Rec}/%		
	Pragma	Private	Reduction	Pragma	Private	Reduction	Pragma	Private	Reduction
PragFormer[16]	0.841	0.924	0.953	0.793	0.716	0.632	0.847	0.663	0.598
GraphCodeBERT(Separated)[17]	0.870	0.937	0.963	0.850	0.768	0.690	0.841	0.684	0.688
OMPify[18]	0.872	0.938	0.966	0.849	0.755	0.690	0.848	0.689	0.700
UniXcoder[19]	0.868	0.935	0.963	0.845	0.794	0.675	0.838	0.649	0.738
CurParCL	0.881	0.940	0.966	0.852	0.800	0.731	0.851	0.723	0.741
CurParCL-X	**0.887**	**0.943**	**0.972**	**0.853**	**0.833**	**0.779**	**0.881**	**0.749**	**0.774**

为进一步验证 CurParCL-X 中各增强模块对并行属性预测任务的贡献，进行了消融实验，结果如表 2 所示。由表 2 可见，基于层次化属性结构，模型整体性能得到提升，尤其在 private 和 reduction 属性预测上表现更加稳定。这说明 pragma、private 和 reduction 并非彼此独立，而是

具有明确的语义依赖关系。将后续子句预测建立在前序属性状态之上，有助于减少不合理属性组合，提高预测结果的结构一致性。动态课程学习进一步提升了模型性能。相比仅按代码长度组织训练样本，本文从代码长度、数据流图(DFG)复杂度和历史误差等因素综合决定样本难度，使模型能够从简单样本逐步过渡到复杂样本。结果表明，该策略有助于缓解复杂数据流和边界样本带来的训练不稳定问题，并提升模型对难判别属性的识别能力。三视图一致性对比学习也带来了明显增益，通过约束原始文本视图、增强文本视图和增强图视图之间的表示一致性，模型能够同时获得更稳健的序列语义表征和更可靠的数据依赖感知能力。其中，文本视图增强提升了模型对代码风格变化的适应性，图视图增强则强化了模型对变量依赖结构的建模能力。错误驱动反馈机制同样有效，将预测错误的假阴性样本作为高风险样本重新纳入后续训练后，模型能够持续修正对边界模式、少数类组合和复杂依赖关系的判别偏差，从而进一步提升预测稳定性。

表2 CurParCL-X框架在并行性检测任务中的消融实验结果

Tab. 2 Ablation experiment results of the CurParCL-X framework on parallelism detection tasks

设置	层次化属性结构	课程学习	图	文本	$F1$/%	R_{Acc}/%
Baseline	No	No	No	No	78.9	90.7
Hierarchical Decoding	Yes	No	No	No	79.3	91.0
Dynamic Curriculum Learning	Yes	Yes	No	No	80.4	91.5
Graph	Yes	Yes	Yes	No	80.7	91.7
Text	Yes	Yes	Yes	Yes	81.4	92.1

消融实验表明，层次化属性结构主要增强属性预测的逻辑一致性，动态课程学习和错误反馈机制提升模型对复杂样本的适应能力，三视图一致性对比学习增强表示鲁棒性。各模块相互补充，共同提升了并行属性预测的性能。

2.4 OpenMP制导语句补全实验结果

在制导语句受控补全实验中，本文选取Qwen2.5-Coder[20]和DeepSeek-Coder[21]作为二阶段代码大模型，用于在prompt指令引导下补全最终OpenMP制导语句。为全面评估所提方法的有效性，本文将其与多类基线方法进行对比，包括仅依赖候选循环代码自由生成的通用大模型，以及面向HPC代码任务进行领域适配的基线方法。表3展示了方法在测试集上的整体性能。

表3 不同模型设置在OpenMP制导语句生成任务中的性能对比

Tab. 3 Performance comparison of different model configurations in OpenMP directive generation tasks

设置	N_{TP}	N_{FP}	N_{TN}	N_{FN}	R_{Pre}/%	R_{Rec}/%	R_{Acc}/%
Qwen2.5-Coder-14B-Instruct(无结构化约束信号)	98	66	121	62	60	61	63
OMPar with ground-truth label	140	57	118	32	71	81	74
Qwen2.5-Coder-14B-Instruct with ground-truth label	120	19	178	30	86	80	85
Deepseek-coder-6.7b-Instruct with ground-truth label	109	30	165	43	78	72	79
OMPar with check	181	16	118	32	92	85	86
Qwen2.5-Coder-14B-Instruct with check	131	8	178	30	94	82	89
Deepseek-coder-6.7b-Instruct with check	112	27	165	43	81	72	80

实验结果表明，引入结构化约束信号后，大模型不再完全依赖代码上下文隐式判断并行语义，而是在受限空间内完成OpenMP制导语句补全。与仅基于候选代码自由生成的方式相比，结构化约束信号为大模型有效缩小了生成空间，减少了无约束生成中常见的负样本误生成和变量幻觉问题。进一步分析发现，以Qwen2.5-Coder在结构化约束输入下的结果为例，在严格gold-label评估下，模型产生了19个假正例，即数据集标注为不具备并行性，但模型生成了OpenMP制导语句，通过人工静态语义复核发现，其中11条循环实际具有较明确的并行化可行性。例如：部分样本属于数组初始化、逐元素赋值、矩阵转置或带有正确reduction子句的累加循环，这些循环虽然在原始数据集中未标注pragma，但从静态语义上看并不存在明显的依赖。因此，这类样本更可能反映原始标签的保守性，而不应简单视为模型语义判断错误。

在相同评测标准下，未引入结构化约束信号的通用大模型表现相对较弱。OMPar集成了经过

OpenMP 任务适配的 MonoCoder-OMP 模型，整体上更倾向于激进生成，能够覆盖更多潜在可并行样本，但也可能带来更高的误生成风险。综上，通过一阶段并行属性预测与二阶段结构化约束驱动补全的结合，使通用代码大模型与领域微调方法之间取得了较好的平衡：既利用结构化约束提升生成可控性，又保留大模型对复杂代码上下文的理解能力。

为验证结构化约束信号各组成部分的作用，本文以仅输入候选循环代码的自由生成方式作为基线，并逐步引入 Decision 和 Clause State、Confidence Scores 以及 Variable Candidates 等约束字段进行消融评估，结果如表 4 所示。

表 4　结构化信号消融实验结果

Tab. 4　Ablation experiment results of structured signal

模型设置	R_{Acc}/%	R_{Pre}/%	R_{Rec}/%	$F1$/%
Code-only LLM	63	60	61	61
＋Decision ＋Clause State	78	77	75	76
＋Confidence Scores	83	85	78	81
＋Variable Candidates	85	86	80	83

由表 4 可知，不提供任何结构化约束信息时，模型缺少明确的决策和变量边界，容易出现负样本误生成、正样本保守跳过以及变量幻觉等问题，因此将其作为性能下界。不同约束字段在补全过程中承担互补作用，并行化决策信号（Decision）用于控制是否触发补全，子句类型状态信号（Clause State）用于限定子句补全状态。属性置信度信号（Confidence Scores）帮助大模型区分强约束与弱约束，尤其可以在边界样本中提供额外的决策依据。变量候选集合（Variable Candidates）用于限制大模型在生成子句变量时的搜索空间，减少不存在变量、类型名误用等变量幻觉问题。完整结构化约束信号通过这些组件的协同作用，使大模型从自由生成转向受控补全，从而提升 OpenMP 制导语句生成的稳定性和可解释性。

2.5　推理性能开销分析

为评估所提两阶段框架在实际推理过程中的性能开销，本文在相同的测试集、硬件平台和生成参数下设置 3 种推理模式进行比较，结果如表 5 所示。

表 5　不同推理模式下的性能开销对比

Tab. 5　Performance overhead comparison under different inference modes

推理模式	第一阶段平均延迟/ms	第二阶段平均延迟/ms	端到端平均延迟/ms	LLM 样本调用率/%	第二阶段吞吐量/(sample•s^{-1})
Code-only LLM	—	1 254.81	1 254.81	100.00	0.797
结构化约束＋全部调用	60.26	1 282.18	1 342.44	100.00	0.780
完整两阶段门控	60.26	591.44	651.70	40.06	1.690

1）Code-only LLM：所有候选循环均直接输入大语言模型，不执行第一阶段并行属性预测，也不提供结构化约束；

2）结构化约束＋全部调用：首先执行 CurParCL-X 并构造结构化约束，但不启用 Skip 门控，所有样本均调用大语言模型；

3）完整两阶段门控：首先由 CurParCL-X 进行并行属性预测，对判定为 Skip 的样本直接终止补全流程，仅对 Generate 样本调用大语言模型。

由表 5 可知，完整两阶段门控仅对 Generate 样本调用大模型，使大模型调用率由 100% 降至 40.06%，减少 59.94 百分点的无效调用。与结构化约束下全部调用相比，端到端平均延迟由 1 342.44 ms 降至 651.70 ms，下降 51.45%，吞吐量提升约 2.17 倍。与此同时，Generate 样本的生成延迟基本不变，说明该门控策略能够在较低额外开销下显著提升整体推理效率。

2.6　局限性与未来工作

尽管本文在统一数据划分和评价指标下选取了多类代表性方法进行比较，但当前实验对比范围仍受到相关方法可复现性的限制。部分近期自动并行化模型尚未公开完整的训练代码、数据预处理流程或关键超参数，难以在严格一致的实验环境下进行公平复现。因此，本文现阶段主要选取具有公开实现、能够按照公开方法复现或与本文任务设置较为接近的代表性模型作为基线，尚未覆盖所有最新自动并行化方法。

后续工作将在相关模型公开完整实现细节后，进一步扩展统一基准下的对比范围，并在相同的数据划分、评价指标和硬件环境下，对不同方法的并行属性识别能力、跨项目泛化性能、推理效率和资源开销进行更加系统的比较。同时，将进一步引入包含复杂指针访问、跨过程依赖和

非规范循环的测试样本，以验证所提方法在复杂真实程序场景中的适用性。

3 结 论

本文提出了一种面向OpenMP自动并行化的结构化约束驱动制导语句补全方法，将串行循环的OpenMP制导语句生成过程解耦为并行属性预测、结构化约束构建和受控制导语句补全3个环节。实验结果表明，CurParCL-X在并行属性预测任务中取得了优于基线模型的性能，能够更有效地捕捉并行属性间的语义依赖关系。完整结构化约束信号能够将大模型的开放式生成空间压缩为补全空间，减少自由生成中常见的负样本误生成、子句误用和变量幻觉问题。与单纯依赖代码上下文的自由生成方式相比，本文的方法将大模型的任务从开放式生成转化为约束条件下的候选指令补全，在不对生成模型进行额外领域微调的情况下提升了OpenMP制导语句补全结果的准确性、稳定性和可解释性。进一步的性能开销实验表明，完整两阶段门控能够减少59.94百分点的大语言模型样本调用，使端到端平均推理延迟下降51.45%，且不会明显增加峰值显存占用，说明该框架具有较好的工程部署潜力，为面向实际应用的自动并行化提供了一条有效路径。

利益冲突声明/Conflict of Interests

所有作者声明不存在利益冲突。

All authors declare no relevant conflict of interests.

参考文献：

[1] Chen L, Mahmud Q I, Phan H, et al. Learning to parallelize with OpenMP by augmented heterogeneous AST representation [PP/OL]. arXiv(2023-05-09)[2026-05-20]. https://arxiv.org/pdf/2305.05779.

[2] Grosser T, Groesslinger A, Lengauer C. Polly: performing polyhedral optimizations on a low-level intermediate representation[J]. Parallel Processing Letters, 2012, 22(4): 1250010.

[3] Li Z, Atre R, Huda Z, et al. Unveiling parallelization opportunities in sequential programs [J]. Journal of Systems and Software, 2016, 117: 282-295.

[4] Shen Y Y, Peng M M, Wang S L, et al. Towards parallelism detection of sequential programs with graph neural network[J]. Future Generation Computer Systems, 2021, 125: 515-525.

[5] Chen L, Mahmud Q I, Jannesari A. Multi-view learning for parallelism discovery of sequential programs[C]// 2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2022: 295-303.

[6] Mahmud Q I, TehraniJamsaz A, Ahmed N K, et al. ConTraPh: contrastive learning for parallelization and performance optimization[C]//Proceedings of the 39th ACM International Conference on Supercomputing, 2025: 596-610.

[7] Bengio Y, Louradour J, Collobert R, et al. Curriculum learning[C]//Proceedings of the 26th Annual International Conference on Machine Learning, 2009: 41-48.

[8] Kadosh T, Hasabnis N, Soundararajan P, et al. OMPar: automatic parallelization with AI-driven source-to-source compilation[PP/OL]. arXiv (2024-09-23)[2026-05-20]. https://doi.org/10.48550/arXiv.2409.14771.

[9] Mahmud Q I, TehraniJamsaz A, Phan H D, et al. AutoParLLM: GNN-guided context generation for zero-shot code parallelization using LLMs[C]//Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, 2025: 11821-11841.

[10] Chen L, Bhattacharjee A, Ahmed N, et al. OMPGPT: a generative pre-trained transformer model for OpenMP [C]//European Conference on Parallel Processing. Cham: Springer Nature Switzerland, 2024: 121-134.

[11] Wang Y, Wang W S, Joty S, et al. CodeT5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation[C]//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021: 8696-8708.

[12] Nichols D, Marathe A, Menon H, et al. HPC-coder: modeling parallel programs using large language models [C]//ISC High Performance 2024 Research Paper Proceedings (39th International Conference), 2024: 1-12.

[13] Etienne N, Garcia de Gonzalo S, Arnold D. OpenMP-annotated code dataset for large language model fine-tuning on parallel programming tasks[J]. Frontiers in High Performance Computing, 2026, 4: 1771927.

[14] Mišić M, Dodović M. An assessment of large language models for OpenMP-based code parallelization: a user perspective[J]. Journal of Big Data, 2024, 11(1): 161.

[15] Nichols D, Davis J H, Xie Z, et al. Can large language models write parallel code? [C]//Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing, 2024: 281-294.

[16] Harel R, Kadosh T, Hasabnis N, et al. PragFormer: data-driven parallel source code classification with transformers[J]. International Journal of Parallel Programming, 2024, 53(1): 2.

[17] Guo D Y, Ren S, Lu S, et al. GraphCodeBERT: pre-training code representations with data flow[C]//9th International Conference on Learning Representations (ICLR), 2021.

[18] Kadosh T, Schneider N, Hasabnis N, et al. Advising OpenMP parallelization *via* a graph-based approach with transformers[C]//OpenMP: Advanced Task-Based, Device and Compiler Programming, 2023: 3-17.

[19] Guo D Y, Lu S, Duan N, et al. UniXcoder: unified cross-modal pre-training for code representation[C]//Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, 2022: 7212-7225.

[20] Hui B, Yang J, Cui Z, et al. Qwen2.5-coder technical report[PP/OL]. V3. arXiv (2024-11-12)[2026-05-20]. https://arxiv.org/abs/2409.12186.

[21] Guo D Y, Zhu Q H, Yang D J, et al. DeepSeek-coder: when the large language model meets programming: the rise of code intelligence[PP/OL]. V2. arXiv (2024-01-26)[2026-05-20]. https://arxiv.org/html/2401.14196.