



2025 AI+ Development
Digital Summit

AI+ 研发数字峰会

拥抱AI 重塑研发

05/23-24 | 上海站



2025 AI+研发数字峰会

拥抱AI 重塑研发 AI+ Development Digital Summit

下一站预告

08/08-09 | 北京站

11/14-15 | 深圳站



查看会议详情

北京站论坛设置

大模型和 AI 应用评测

智能存储与检索技术

下一代知识工程

AI+ 金融业务创新

智能需求工程

智能体与研发效率工具

AI 产品运营与出海策略

大模型安全与对齐

大模型应用开发框架与实践

智能体经济 (Agentic Economy)

智能测试工具的开发与应用

具身智能与机器人

代码生成及其改进

AI+ 新能源汽车

AI 前沿技术探索与实践

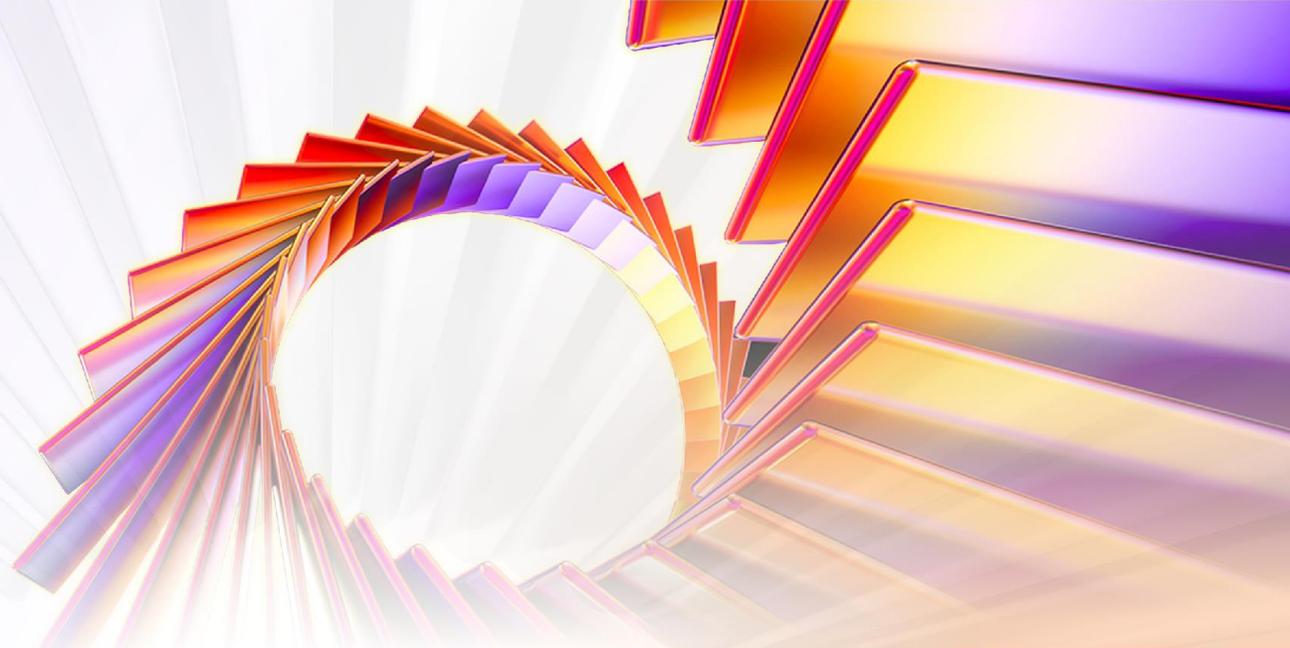


| 05/23-24 | 上海站

2025 AI+ Development
Digital Summit

AI+研发数字峰会

拥抱AI 重塑研发



NASL领域大模型训练

刘东 | 网易数智



刘东

网易 AI算法专家

2012年毕业于清华大学电子工程系，长期从事人工智能领域算法与系统的研发工作，参与过OCR、语音识别、机器翻译、机器学习平台等多个方向的项目研发，目前主要负责网易CodeWave产品中NASL大语言模型以及多模态模型的相关研发工作。

目录

CONTENTS

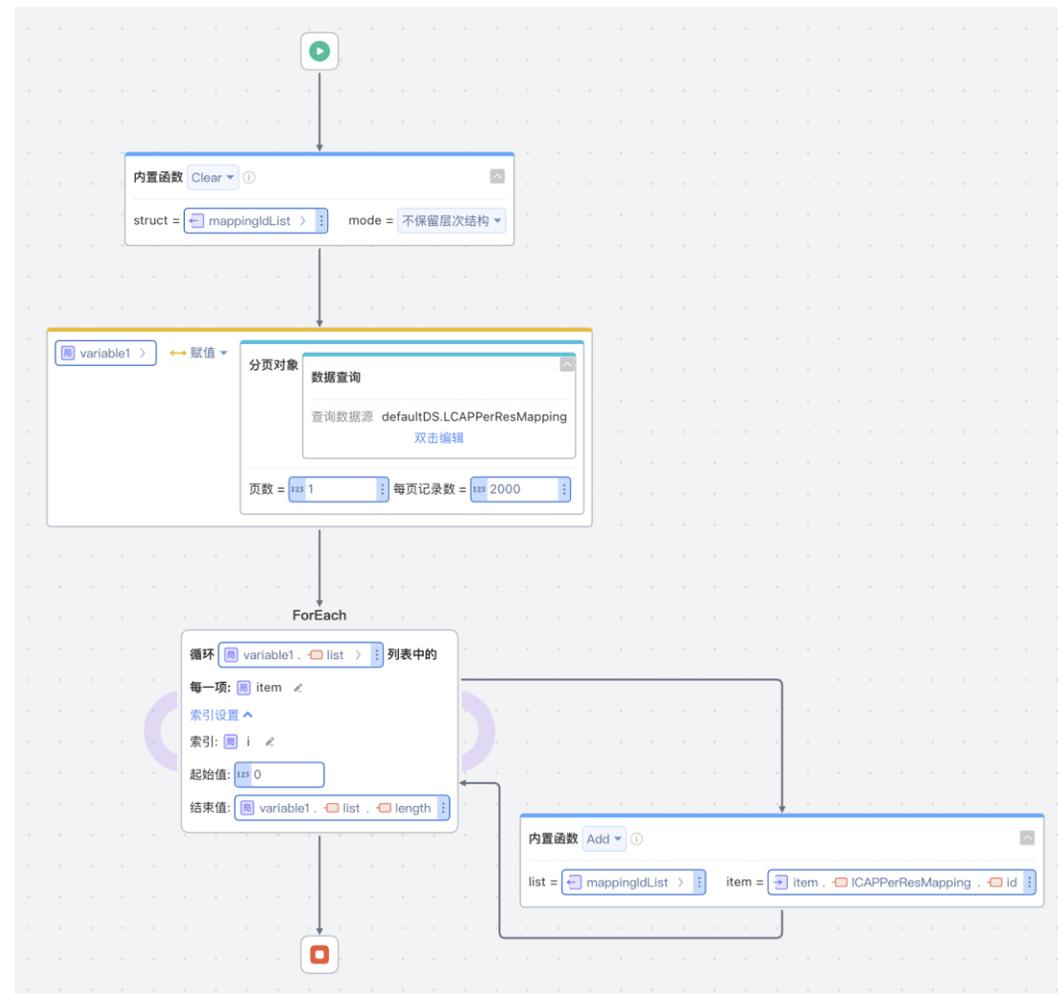
- I. CodeWave智能开发平台与NASL语言介绍
- II. NASL领域模型训练项目背景
- III. NASL领域模型训练整体思路
- IV. NASL领域数据构造挑战与解决方案
- V. 模型训练与评测
- VI. 总结与展望

PART 01

CodeWave智能开发平台 与NASL语言介绍

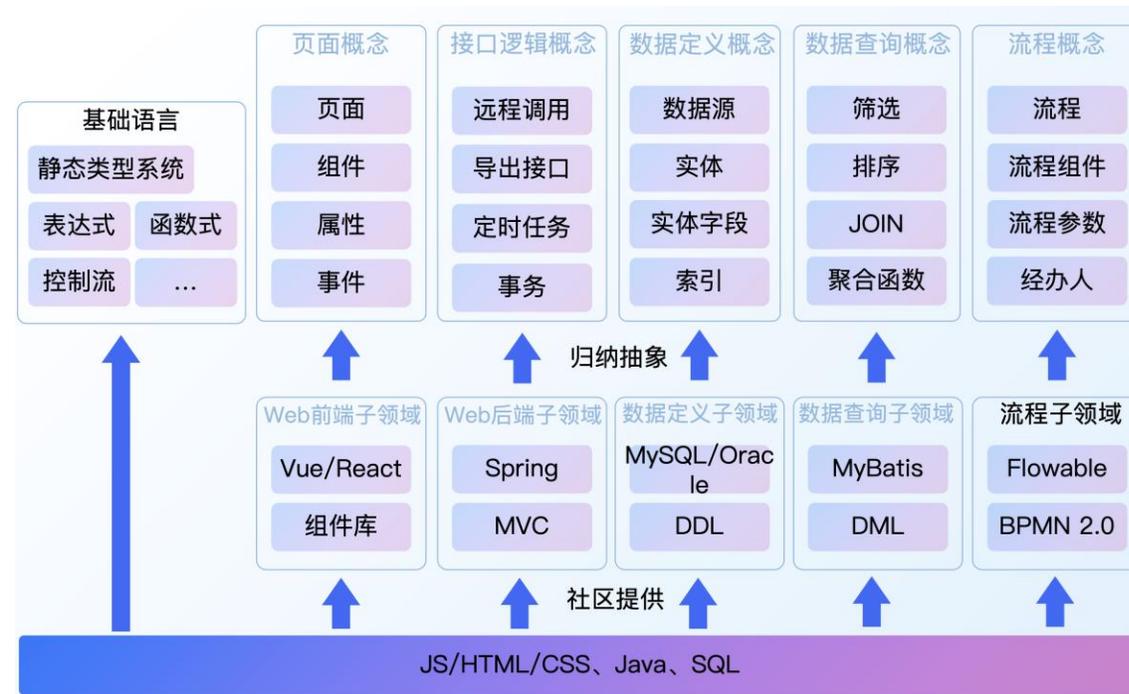
CodeWave智能开发平台介绍

- 网易数智旗下低代码产品
- 提供可视化的开发环境
- 通过图形化界面、拖拽组件、配置参数等方式来创建应用程序
- 无需编写大量的传统代码



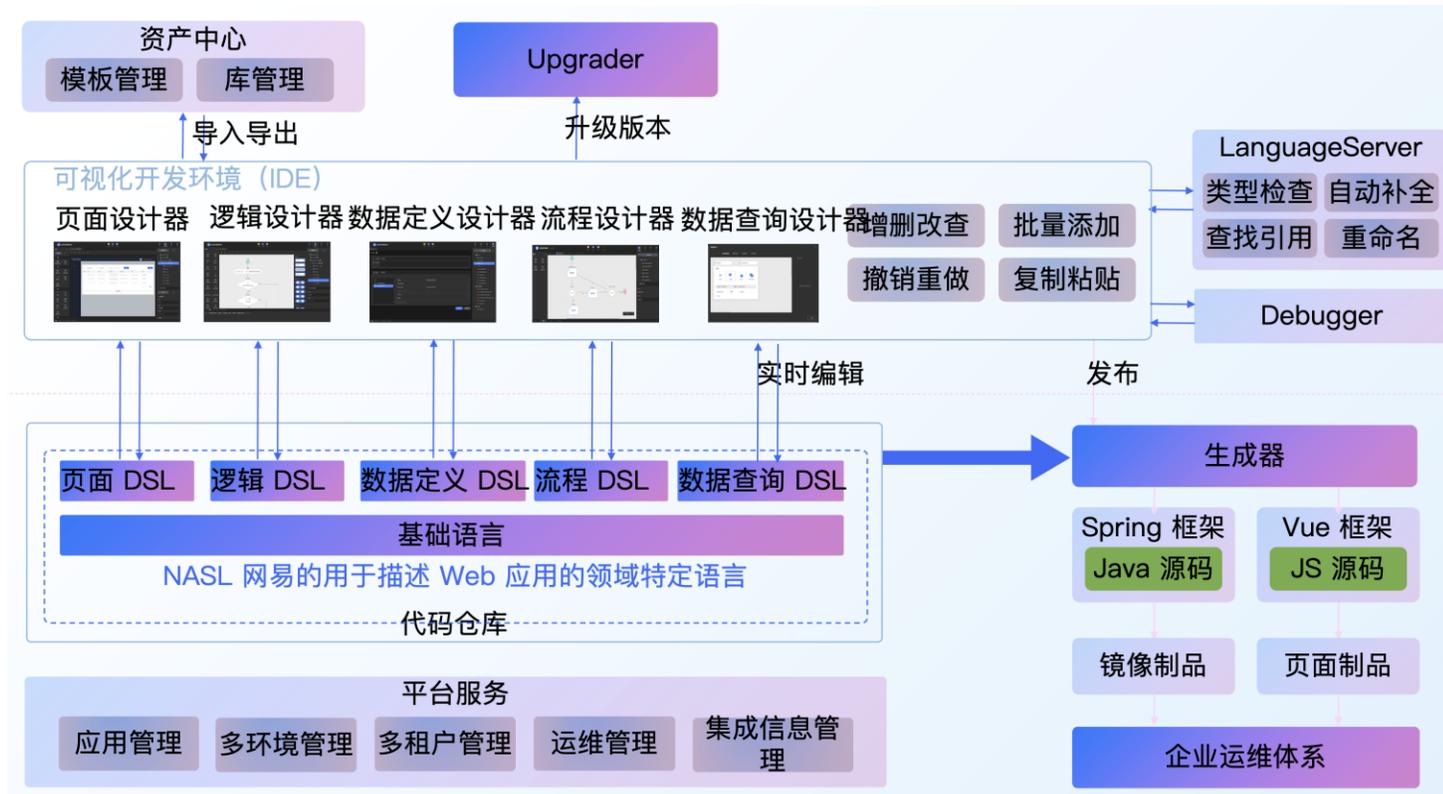
CodeWave相对传统开发的优势

- 传统应用开发概念多，技术栈复杂，对开发人员要求高
- 低代码产品降低用户门槛，没有深厚编程背景的业务人员也可使用



CodeWave整体技术方案

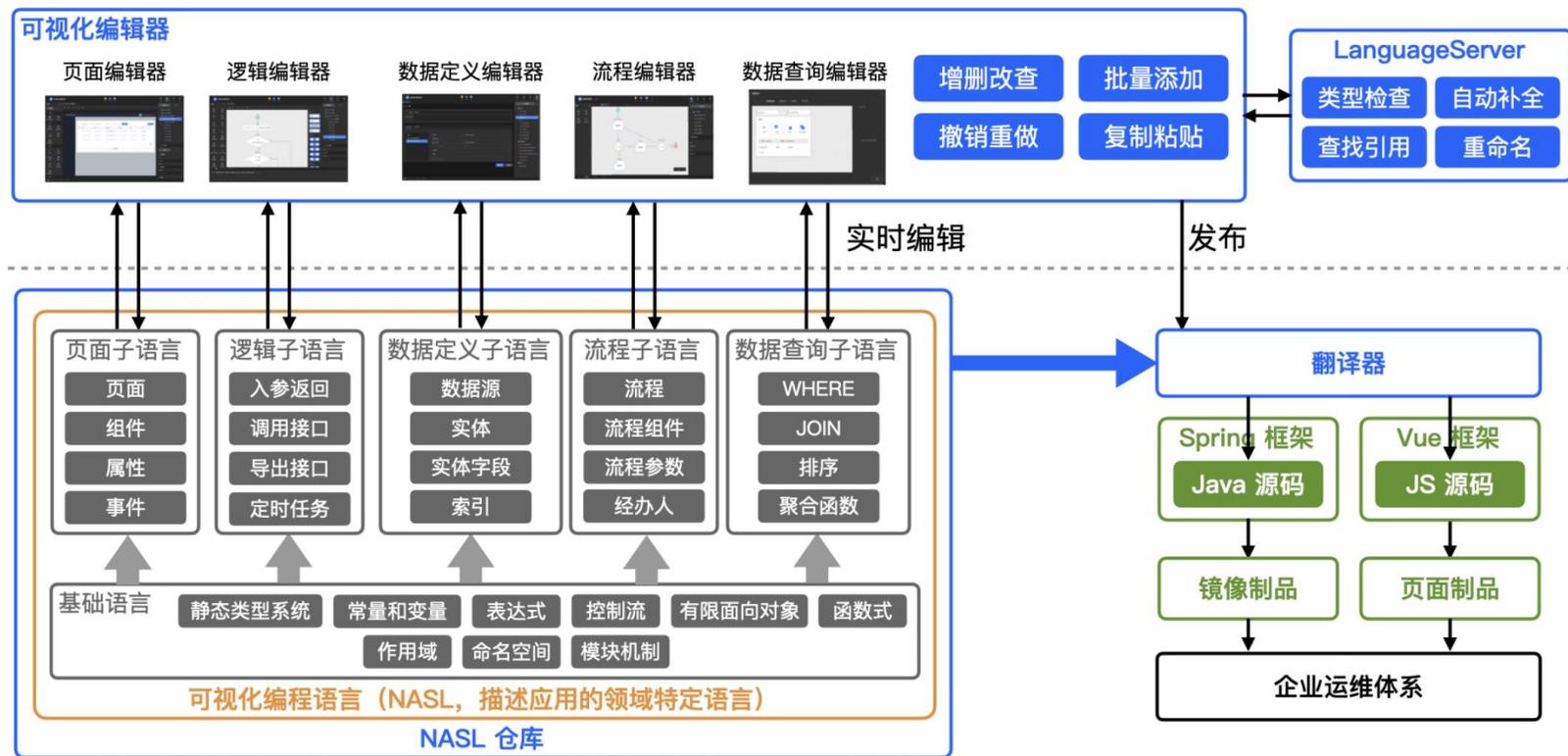
- 以统一的NASL语言为核心
- 提供完善的工具链
- 用户在IDE中的可视化编辑，本质是在写NASL代码



CodeWave核心NASL语言

- Next Application Specific Language

- 基础语言
- 特定领域子语言



▶ CodeWave面临的挑战与思路

- 用户的顾虑
 - 有学习成本
 - 开发效率、开发质量能否提升
- 解决思路
 - 引入基于AI的copilot工具辅助用户
 - 自然语言写代码、代码补全
- 算法需求：底层需要懂NASL语言的大模型驱动



PART 02

NASL领域模型训练项目背景

用CodeWave的NASL语言写计算编辑距离

• 现有大模型均不了解NASL语言

用CodeWave低代码平台的NASL语言，写一段计算编辑距离的代码

📌 编辑距离 (Levenshtein距离) 计算 (CodeWave NASL语言)

```
nasl
// 计算两个字符串之间的编辑距离
function levenshteinDistance(str1, str2) {
  let m = length(str1);
  let n = length(str2);

  // 创建一个(m+1) x (n+1)的矩阵
  let dp = createArray(m + 1, n + 1);

  // 初始化第一行和第一列
  for(let i = 0; i <= m; i++) {
    dp[i][0] = i;
  }
  for(let j = 0; j <= n; j++) {
    dp[0][j] = j;
  }

  // 动态规划填充矩阵
  for(let i = 1; i <= m; i++) {
    for(let j = 1; j <= n; j++) {
      // 如果字符相同, 则不需要操作
      if(str1[i-1] == str2[j-1]) {
        dp[i][j] = dp[i-1][j-1];
      } else {
        // 否则取插入、删除、替换中的最小值加1
        dp[i][j] = min(
          dp[i-1][j] + 1, // 删除
          dp[i][j-1] + 1, // 插入
          dp[i-1][j-1] + 1 // 替换
        );
      }
    }
  }
}
```

注意：此代码假设CodeWave NASL语言支持二维数组和基本的数组操作。如果实际平台语法有所不同，可能需要进行相应调整。

由于 CodeWave 的 NASL 语言并没有广泛公开的资料，下面我按照通用的编程逻辑给出一个计算编辑距离 (Levenshtein 距离) 的代码示例，假设 CodeWave 的 NASL 语言有类似常见编程语言的基本语法结构。编辑距离指的是两个字符串之间，由一个字符串转换成另一个字符串所需的最少编辑操作 (插入、删除、替换) 次数。

```
plaintext ^
# 计算编辑距离的函数
function editDistance(str1, str2) {
  let m = str1.length;
  let n = str2.length;

  # 创建一个二维数组来存储子问题的解
  let dp = [];
  for (let i = 0; i <= m; i = i + 1) {
    dp[i] = [];
    for (let j = 0; j <= n; j = j + 1) {
      if (i === 0) {
        dp[i][j] = j;
      } else if (j === 0) {
        dp[i][j] = i;
      } else if (str1[i - 1] === str2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1];
      } else {
        dp[i][j] = 1 + Math.min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]);
      }
    }
  }

  return dp[m][n];
}
```



▶ NASL领域模型训练—prompt方案

- 通过优化提示词的方式，有一定效果



```
function logic1(a: String, b: String) {  
    let lenA: Long = nasl.util.Length(a);  
    let lenB: Long = nasl.util.Length(b);  
    let dp: List<Long> = nasl.util.NewList<Long><[]>;  
  
    // 初始化dp数组  
    ForEach(nasl.util.ListRange(0, lenB + 1), 0, lenB + 1, (i) => {  
        nasl.util.ListAdd(dp, i);  
    });  
  
    ForEach(nasl.util.ListRange(1, lenA + 1), 0, lenA, (i) => {  
        let current: List<Long> = nasl.util.NewList<Long>([i]);  
        ForEach(nasl.util.ListRange(1, lenB + 1), 0, lenB, (j) => {  
            let cost: Long = (nasl.util.SubString(a, i - 1, 1) == nasl.util.SubString(b, j  
            let minVal: Long = nasl.util.ListGet(dp, j - 1) + cost;  
            let val1: Long = nasl.util.ListGet(dp, j) + 1;  
            let val2: Long = nasl.util.ListGet(current, j - 1) + 1;  
            if (val1 < minVal) {  
                minVal = val1;  
            }  
            if (val2 < minVal) {  
                minVal = val2;  
            }  
            nasl.util.ListAdd(current, minVal);  
        });  
        dp = current;  
    });  
  
    return nasl.util.ListGet(dp, lenB);  
}
```



▶ NASL领域模型训练—prompt方案评测

- 基础语言部分—HumanEval测试集
- 人工排除与web开发完全无关的问题，剩余问题改写为中文

模型	通过率
gpt-4o-mini	33.14%
gpt-4o	55.86%
deepseek-V3	40.86%
deepseek-R1	44.29%
deepseek-70b	29.00%
claude-3.7	60.71%



▶ NASL领域模型训练—prompt方案评测

- 相比其它主流语言还有很大的提升空间
- 主要表现是幻觉严重，语法错误多
- 本质上是考验大模型的指令遵循能力
- prompt需要持续增加要求但大模型无法完全遵守

大类	影响分数	占比	能提升的分数幅度	问题分类	分数-类	能提升的分数幅度
语言能力（正在进行中）	29.2	23.3%	10.4%	不支持for(let i)	7.3	2.6%
				内置函数不是通用姿势	13.7	4.9%
				无法构建range列表	3.9	1.4%
				无法构建n长度的空列表	2.9	1.0%
				多字段排序	1.4	0.5%
语言能力和内置函数（待开始）	42.9	34.2%	15.3%	局部作用域函数	15.1	5.4%
				不支持break/continue	7.4	2.6%
				多行 lambda	1.0	0.4%
				优先返回null的类型问题	1.3	0.5%
				数学库	10.4	3.7%
				正则表达式	2.0	0.7%
				内置函数能力不足	5.6	2.0%
需要 AI 训练	51.1	40.8%	18.3%	数值类型合并问题	6.6	2.4%
				转换问题	1.4	0.5%
				不支持for(let i)	9.7	3.5%
				未处理ForEach参数重名的问题	0.8	0.3%
				Infinity	2.8	1.0%
				类型判断的问题	3.0	1.1%
其他	2	1.6%	0.7%	AI理解有差距	26.8	9.6%
				题目不合理	2.0	0.7%
合计	125.2	100%	44.7%			
	280					

早期使用 HumanEval Benchmark 对生成 NASL 逻辑的评测



▶ NASL领域模型参数训练的必要性

- 自行训练NASL领域模型相比采用通用模型有优势

	NASL领域模型(7/14B)	通用模型(gpt/claude/deepseek)
生成效果	好	一般
生成速度	快	慢
私有化部署	支持	除deepseek外均不支持
部署成本	低, 单GPU卡	高, GPU集群

- CodeWave以私有化部署交付的模式需要NASL领域模型的支撑

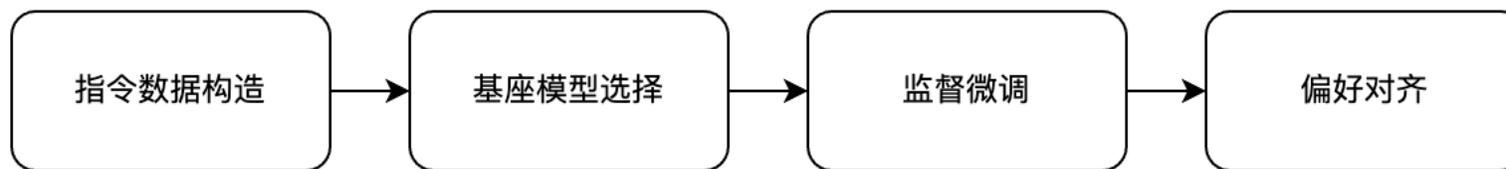


PART 03

NASL领域模型训练整体思路

▶ NASL领域模型训练整体思路

- 直接在对话模型基础上做指令微调



- 先从基础语言部分入手，再逐步扩展逻辑、实体定义、数据查询等子语言领域



▶▶ 不做预训练的考虑

- 缺少大量的高质量NASL代码库，数据获取成本太高
- 选择大模型更加熟悉的Natural TS代码作为NASL的表达方式

文本化 NASL Natural TS

```
@(
  title = "标题",
  description = "",
  triggerType = "",
  cron = "",
  transactional = false,
  overridable = false,
)
logic logic1(list: List<Integer>) => result {
  result = null
  if (list.length == 0) {
    result = null
  } else {
    result = ListSum(list) / list.length
  }
}
end
}
```

文本化 NASL Natural TS

```
function logic1(list: List<Long>) {
  let result;
  result = null
  if (list.length == 0) {
    result = null
  } else {
    result = nasl.util.ListSum(list) / list.length
  }
  return result;
}
```



▶ 基座模型选择

- 持续跟踪和评测开源模型
- 主要使用了DeepSeek-Coder、Qwen-Coder
- 基座模型性能接近，决定最终效果的核心因素是数据的质量

模型	链接
DS-Coder-1.3B-Base	https://hf.co/deepseek-ai/deepseek-coder-1.3b-base
DS-Coder-6.7B-Base	https://hf.co/deepseek-ai/deepseek-coder-6.7b-base
DS-Coder-33B-Base	https://hf.co/deepseek-ai/deepseek-coder-33b-base
DS-Coder-V2-Lite-Base	https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Lite-Base
DS-Coder-V2-Base	https://hf.co/deepseek-ai/DeepSeek-Coder-V2-Base
Qwen2.5-Coder-0.5B	https://hf.co/Qwen/Qwen2.5-Coder-0.5B
Qwen2.5-Coder-1.5B	https://hf.co/Qwen/Qwen2.5-Coder-1.5B
Qwen2.5-Coder-3B	https://hf.co/Qwen/Qwen2.5-Coder-3B
Qwen2.5-Coder-7B	https://hf.co/Qwen/Qwen2.5-Coder-7B
Qwen2.5-Coder-14B	https://hf.co/Qwen/Qwen2.5-Coder-14B
Qwen2.5-Coder-32B	https://hf.co/Qwen/Qwen2.5-Coder-32B
CodeQwen1.5-7B	https://hf.co/Qwen/CodeQwen1.5-7B
StarCoder2-3B	https://hf.co/bigcode/starcoder2-3b
StarCoder2-7B	https://hf.co/bigcode/starcoder2-7b
StarCoder2-15B	https://hf.co/bigcode/starcoder2-15b



PART 04

NASL领域数据构造挑战 与解决方案

▶ NASL领域数据构造—目标数据

- <用户指令、项目上下文、NASL代码>的形式
- 项目上下文是可选项

```
1 // 用户指令：接受一个字符串列表，并返回最短的字符串。如果有多个，返回第一个出现的字符串。
2 // 合成的NASL代码
3 function findShortest(strings: List<String>) {
4     if (nasl.util.Length(strings) == 0) {
5         return '';
6     }
7     let lengths: List<Long> = nasl.util.ListTransform(strings, (item) =>
nasl.util.Length(item));
8     let minLength: Long = nasl.util.ListMin(lengths);
9     let shortestString: String = nasl.util.ListFind(strings, (item) =>
nasl.util.Length(item) == minLength);
10    return shortestString;
11 }
```



▶ NASL领域数据构造—指令收集

- 主要从开源数据中收集原始指令，如opencoder、wikisql等
- 对原始指令做中文改写以及质量过滤
 - 和CodeWave产品相关性
 - 指令的难度分布
- 对于开源数据不覆盖的场景，人工借助llm做构造
 - 部分nasl函数的特定使用方式
 - 包含项目上下文的场景等
- 大部分指令包含对应的测试用例



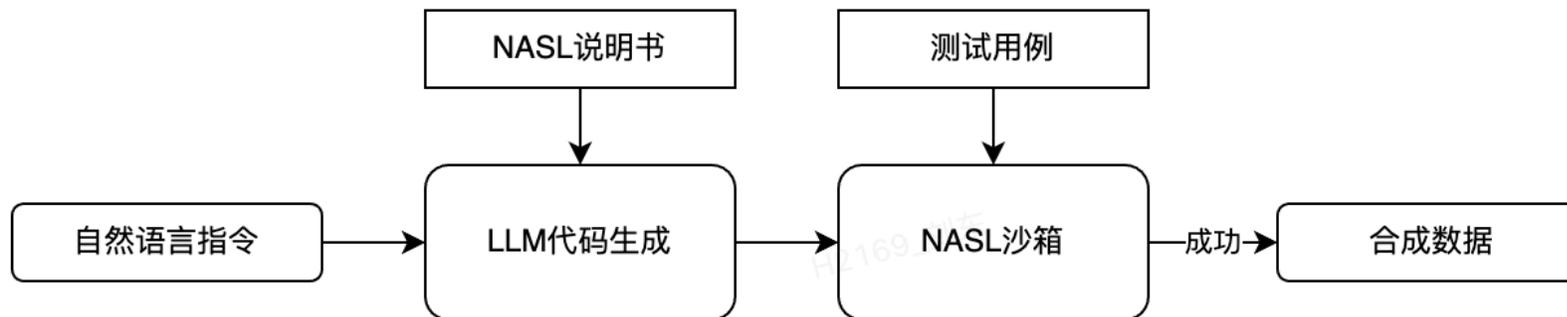
- CodeWave语言实验室提供轻量化NASL语言沙箱

- 语言编译：
 - 语法正确性判断
 - 静态类型检查
 - 最佳实践检查
- 代码执行：
 - 环境隔离
 - 并行执行
 - 系统和浏览器环境模拟
 - 数据库和外部依赖模拟
- 结果比对



▶ NASL领域数据构造—代码合成的挑战

- 利用LLM基于语言说明书生成代码

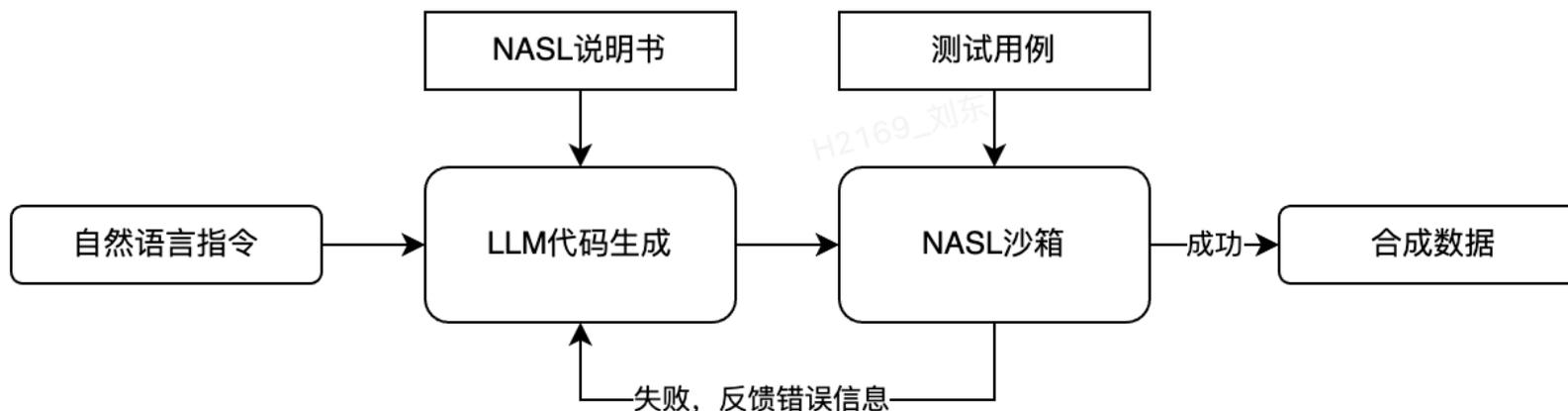


- 合成成功率低(基础语言部分约30%), 主要以简单问题为主
- 困难问题难以合成



▶ NASL领域数据构造—合成方案改进

- 引入基于错误反馈的多轮合成机制



- NASL沙箱针对性改进，提供错误信息以及建议的修复方案
- 合成成功率改进显著(基础语言部分60%+)



▶ NASL领域数据构造—多轮反馈示例

- 用户指令：根据申请状态查询请假申请记录，支持分页，按申请时间升序排列

```
// 第1次生成      // 第2次生成      // 第3次生成      // 第4次生成
function queryL function queryLea function queryLe function queryLeaveRequests(applyStatus: app.enums.ApplyStatus, page: Long, size: Long) {
    let result: OM(app.dataSour OM(app.dataSource OM(app.dataSourc let result: { list: List<{ leaveRequest: app.dataSources.defaultDS.entities.LeaveReque
    .WHERE(      .WHERE(Le      .WHERE(L      st }>; total: Long; } = PAGINATE(
    .ORDER_      .ORDER_BY      .SELECT(      FROM(app.dataSources.defaultDS.entities.LeaveRequestEntity, LeaveRequest => $(
    );          .SELECT({      .WHERE(LeaveRequest.applyStatus == applyStatus)
    return PAGI      leave      })          .SELECT({
    }            }          leaveRequest: LeaveRequest
    // NASL沙箱编译报      );          .ORDER_BY((resultItem) => [[LeaveRequest.createdTime, 'ASC']])
    }            }          ),
    // NASL沙箱编译报错 // NASL沙箱类型检      );          page,
    > SELECT/AGGR_SEL { list: List<{ l      return PAGIN      size
    List<{ leaveRequ }          }          );
    // NASL沙箱编译成功，类型检查成功，合成结束。
```



▶ NASL领域数据构造—推理模型引入

```
// 用户指令：统计每个月创建的合同个数
// 4o-mini经过5轮修复后的结果，但仍然编译有问题
function getIndexContractTrending() {
  let contractCountByMonth: List<{ month: String, count: Long }> =
  FROM(app.dataSources.defaultDS.entities.ContractEntity, Contract => $()
    .GROUP_BY(nasl.util.FormatDateTime(Contract.createdTime, 'yyyy-MM')) // 按年月分组
    .AGGR_SELECT({
      month: nasl.util.FormatDateTime(Contract.createdTime, 'yyyy-MM'), // 格式化为'yyyy-MM'
      count: COUNT(Contract.id) // 统计合同数量
    })
  );
  return contractCountByMonth;
}
```

r1的思考过程

首先，我需要使用NASL的数据查询语法，特别是FROM函数来查询合同表。

...

根据项目上下文中的定义，用户可能需要使用nasl.util.FormatDateTime函数，但该函数是否可以在数据查询中使用？根据限制，查询链中禁止使用nasl.util的函数，所以只能在应用层处理。

可能的解决方案： 1. 查询所有合同的createTime。2. 历每个合同，将createTime格式化为'yyyy-MM'字符串。3. 按字符串分组统计数量。4. 按月份排序。

...

• 显著提升复杂问题的成功率

	非深度思考模型 (gpt-4o/4o-mini)	DeepSeek-R1
1次生成成功率	0%	20%
5轮修改成功率	0%	60%



▶ NASL领域数据构造—后处理

- 规则过滤
 - 结合训练评测中的反馈，快速过滤掉数据中典型的问题
- 数据组装
 - 根据训练需求，组织训练数据



PART 05

模型训练与评测

▶▶ 模型训练—监督微调&偏好对齐

- 基于lora的方式做训练
- SFT：由粗到细的训练方式
 - 基于初步训练结果做拒绝采样
 - 利用拒绝采样结果继续训练
- 偏好对齐：采用DPO方案
 - 基于SFT结果，通过采样获得正负样例
 - 训练后困难问题的生成效果有提升



▶ 模型训练—结果评测

	GPT4o	NASL领域模型（14B）
HumanEval	55.86	78.57
内部测试集1	68.81	83.05
内部测试集2	56.67	76.66
内部测试集3	47.82	56.62



PART 06

总结与展望

- 针对性的构造NASL领域数据并进行模型微调，显著提升了特定领域的效果
- 当前针对困难问题的生成能力还有较大提升空间，还需进一步探索降低幻觉的方案



参与调研您将优先获得



AiDD定制版
《AI+软件研发精选案例》



专属学习顾问
1对1需求对接

AiDD会后小调研

AiDD峰会致力于协助企业利用AI技术深化计算机对现实世界的理解，推动研发进入智能化和数字化的新时代。作为峰会的重要共建者，您的真知灼见对我们至关重要。衷心感谢您的参与支持！

2025 AI+研发数字峰会

拥抱 AI 重塑研发



扫码参与调研

科技生态圈峰会 + 深度研习

—1000+ 技术团队的选择



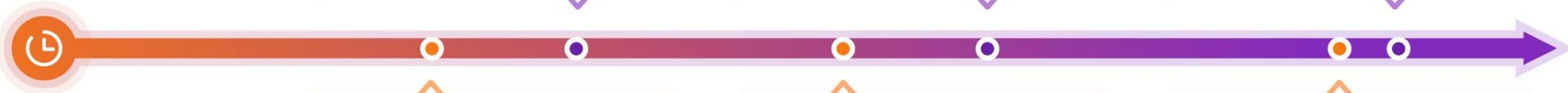
K+峰会 **敦煌站**
K+ 思考周®研习社
时间: 2025.08.29-30

K+峰会 **上海站**
K+ 金融专场
时间: 2025.09.26-27

K+峰会 **香港站**
K+ 思考周®研习社
时间: 2025.11.17-18



K+峰会详情



AIDD峰会 **上海站**
AI+研发数字峰会
时间: 2025.05.23-24

AIDD峰会 **北京站**
AI+研发数字峰会
时间: 2025.08.08-09

AIDD峰会 **深圳站**
AI+研发数字峰会
时间: 2025.11.14-15



AIDD峰会详情



2025 AI+研发数字峰会
AI+ Development Digital Summit

感谢聆听!

扫码领取会议PPT资料

