AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

AI驱动软件研发
全面进入数字化时代

中国·深圳 11.24-25

AI+
software
Development
Digital
summit

大模型驱动的自动化日志分析

贺品嘉
香港中文大学（深圳）数据科学学院

科技生态圈峰会 + 深度研习

——1000 + 技术团队的共同选择

KEYLINK ing

K+峰会

K+峰会 深圳站

K+ 全球软件研发行业创新峰会

会议时间：2024.05.24-25

K+峰会 上海站

K+ 全球软件研发行业创新峰会

会议时间：2024.09.20-21

AiDD 峰会

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2023.11.24-25

AiDD峰会 北京站

AI+ 软件研发数字峰会

会议时间：2024.07.19-20

AiDD峰会 深圳站

AI+ 软件研发数字峰会

会议时间：2024.11.15-16

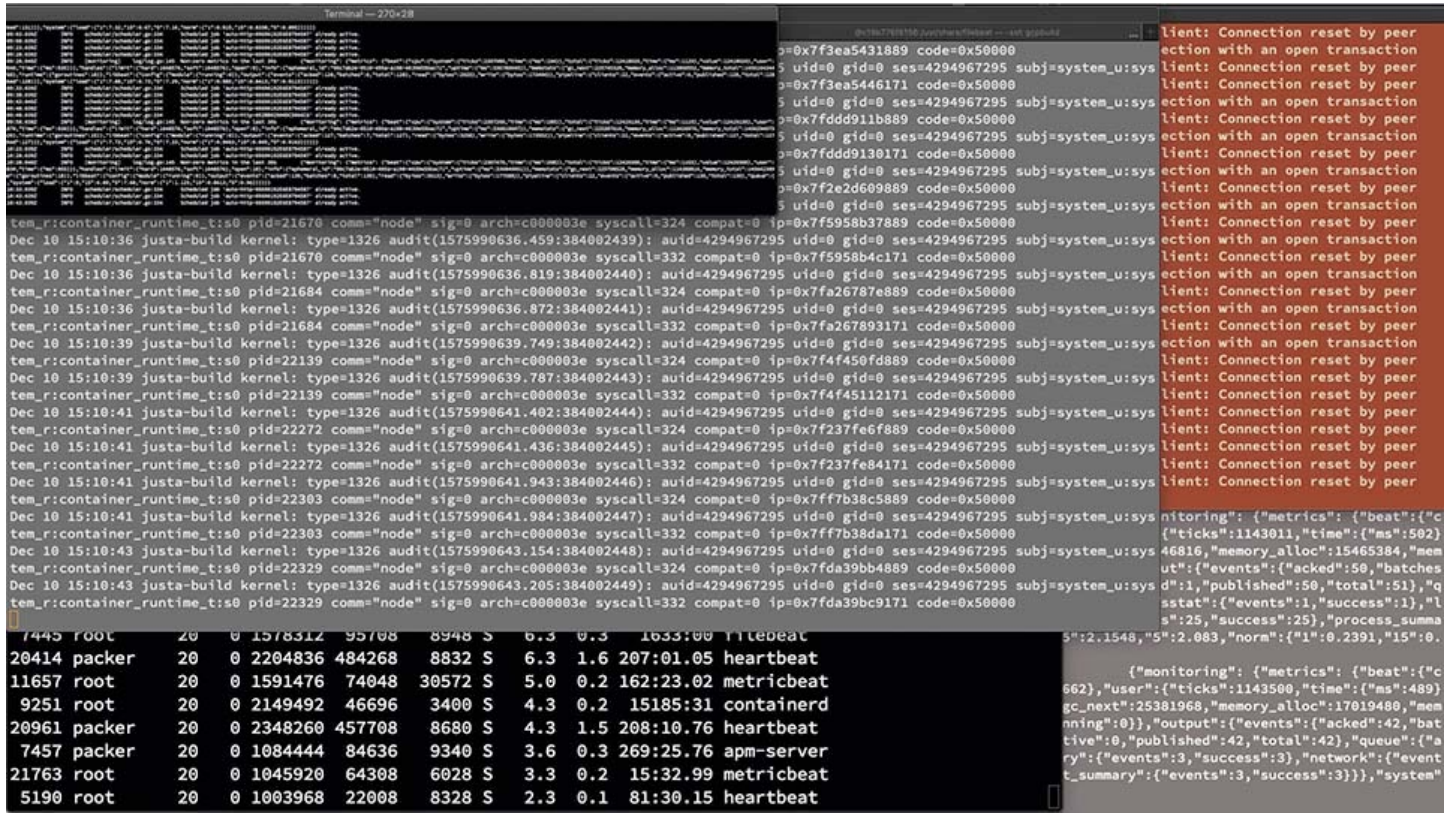
目录
CONTENTS

PART 01
传统日志分析：发展概述

日志分析

1. 如何记录高质量的规范日志?
2. 如何管理和保存大量日志?
3. 如何从日志中提取重要信息?
4. 如何利用日志中挖掘到的信息?

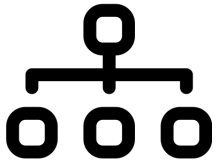
为探究并解决这些问题，自动化日志分析应运而生。



日志分析

日志记录

日志压缩



日志解析

日志挖掘

通常认为广义的自动化日志分析由如上四个阶段组成，以解决海量日志场景下，日志的生成、收集、管理、利用等问题。

日志分析

日志记录

- 关注两种与错误相关的代码段：
  - Exception snippets: try-catch blocks
  - Return-value-check snippets: function-return errors

例子 1

```
try {
        method(…);
}
catch (IOException)
{
        log(…);
        …
}
```

例子 2

```
var res =
method(…);
if (res == null) {
        log(…);
        …
}
```

日志分析

日志解析

解析前 2008-11-11 03:41:48 Received block blk_90 src:

/10.251.30.6 dest: /10.251.30.6: of size 67108864

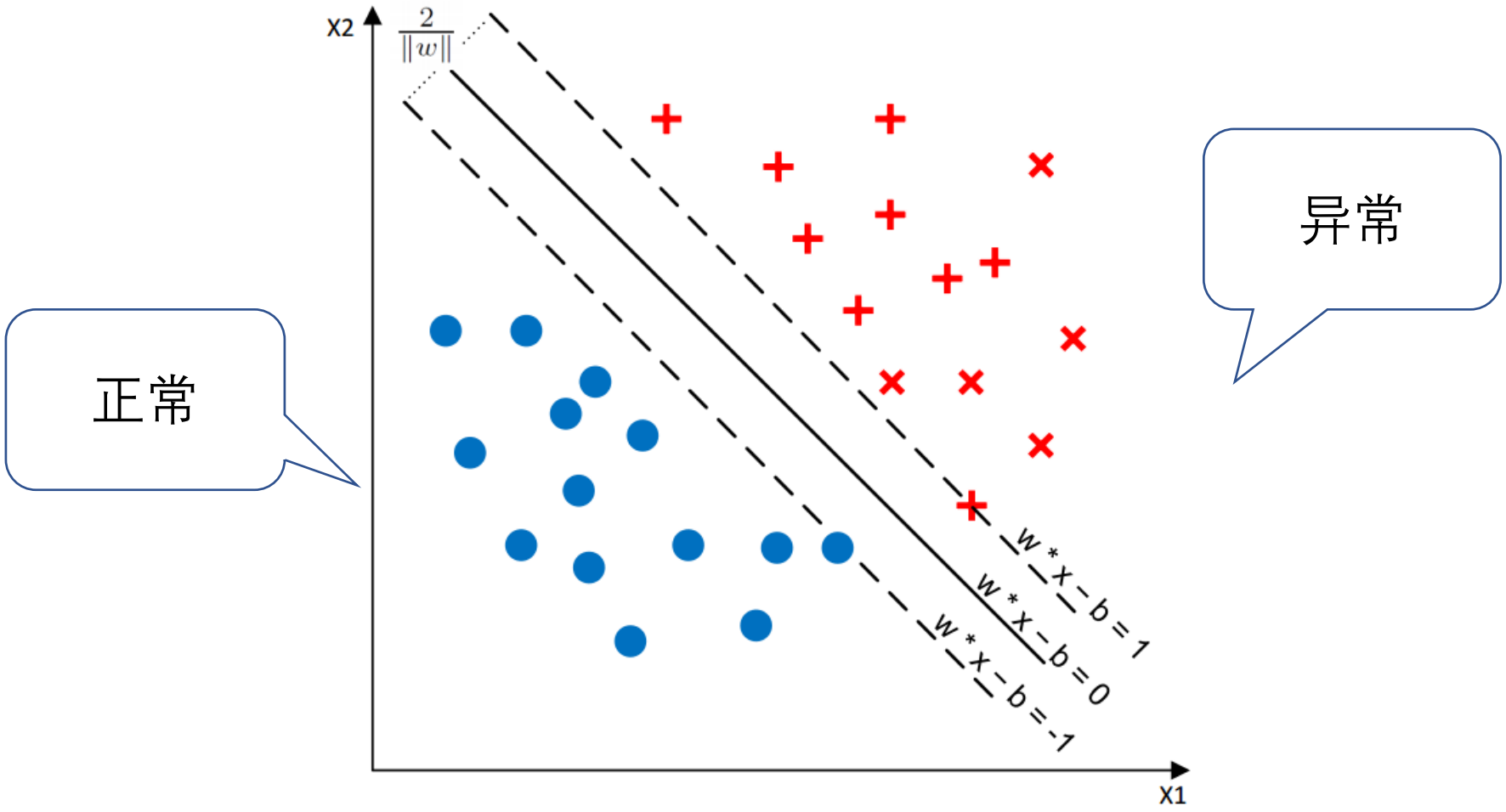
Log Parsing

```
LOG.info("Received block " + block +
         " src: " + remoteAddress +
         " dest: " + localAddress +
         " of size " + block.getNumBytes());
```

解析后 blk_90 -> Received block * src: * dest: * of size *

日志分析

日志挖掘



PART 02

传统日志分析：局限与挑战

传统日志分析的挑战：日志记录

1. 难以端到端地完成日志记录任务

日志记录的每个细分子任务之间差异很大，例如：

- 日志等级的预测可以建模为一个多分类问题；
- 日志消息的预测可以建模为一个文本生成问题；
- 日志语句的位置的预测则有多种问题建模方式。

因此，传统日志记录方法通常局限于单个子任务，而难以实现统一的端到端日志记录。

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     client.flush();
5|     client.close().addListener(CLOSE);
6|}
```

子任务#1: 在合适的位置处插入日志语句: Line#4

子任务#2: 设置合适的日志等级: LOG.Info()

子任务#3: 记录合适的日志消息: "Disconnect successful, clientID: {}"

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     LOG.info("Disconnect successful, clientID: {}", clientID);
5|     client.flush();
6|     client.close().addListener(CLOSE);
7|}
```

传统日志分析的挑战：日志记录

2. 难以细粒度完成日志记录

实际工业场景下，业务代码的结构和风格千变万化，导致传统方法难以精准捕捉到日志语句和业务代码的关联特征：

- 难以判断业务代码需要的日志语句的数量和确切的位置。
- 难以精确判断日志消息的模版和需要记录的变量。

因此，细粒度的日志记录难以通过传统日志分析方法实现。导致其难以被用于实际软件开发环境中。

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     client.flush();
5|     client.close().addListener(CLOSE);
6|}
```

子任务#1: 在合适的位置处插入日志语句: Line#4

子任务#2: 设置合适的日志等级: LOG.Info()

子任务#3: 记录合适的日志消息: "Disconnect successful, clientID: {}"

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     LOG.info("Disconnect successful, clientID: {}", clientID);
5|     client.flush();
6|     client.close().addListener(CLOSE);
7|}
```

传统日志分析的挑战：日志记录

3. 受限于方法本身的约束

许多传统方法需要比较多的已知信息才能正常工作。因此，这些方法在设计之初就假设了一些强假设的场景，例如：

- 仅能根据日志消息和源码上下文推荐日志等级。
- 仅能在确定位置上，根据源码上下文来推荐日志消息。

但是，这些强假设的场景在开发中几乎无法遇见，导致传统方法难以实用。

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     client.flush();
5|     client.close().addListener(CLOSE);
6|}
```

子任务#1: 在合适的位置处插入日志语句: Line#4

子任务#2: 设置合适的日志等级: LOG.Info()

子任务#3: 记录合适的日志消息: "Disconnect successful, clientID: {}"

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     LOG.info("Disconnect successful, clientID: {}", clientID);
5|     client.flush();
6|     client.close().addListener(CLOSE);
7|}
```

传统日志分析的挑战：日志解析

1. 难以跨日志系统泛化

不同的日志系统之间的日志消息的结构和风格千差万别。对于传统方法而言，日志模版在不同日志系统中的特征难以被抽象和捕捉到。例如：

- 在部分日志系统中，“Close”和“Open”可能是作为模版的一部分出现。它们的模版对应不同的事件名。
- 在部分日志系统中，“Close”和“Open”可能是作为参数的一部分出现。它们分别代表同一事件中的不同变量值。

由于传统日志解析方法无法结合整体语义进行解析，因此在跨系统泛化上表现较差。

日志

```
2023-01-14 23:05:14 INFO:  Reading data from /user/input/file.txt
2023-01-14 23:05:14 DEBUG: Setting block size to 1919810
2023-01-14 23:05:14 INFO:  Setting replication factor to 4
2023-01-14 23:05:14 ERROR: /user/input/file.txt does not exist
```

结构化数据

Date	Time	Level	Template	Parameters
2023/1/14	23:05:14	INFO	Reading data from <*>	['/user/input/file.txt']
2023/1/14	23:05:14	DEBUG	Setting block size to <*>	['1919810']
2023/1/14	23:05:14	INFO	Setting replication factor to <*>	['4']
2023/1/14	23:05:14	ERROR	<*> does not exist	['/user/input/file.txt']

传统日志分析的挑战：日志解析

2. 难以解析混合有多行消息和表格消息的日志

包含有多行消息和表格消息的日志的结构复杂性较大，导致现有方法难以精确解析这类日志数据。[1]

```
[17:50:30] mkdir: cannot create directory '/home/team7/results': File exists
[17:50:30] mkdir: cannot create directory '/home/team7/results/terasort': File exists
[17:50:30] mkdir: cannot create directory '/home/team7/results/bayes': File exists
[17:50:30] mkdir: cannot create directory '/home/team7/results/pagerank': File exists
[17:50:31] Traceback (most recent call last):
  File "/HiBench3/bin/functions/load_config.py", line 713, in <module>
    load_config(conf_root, workload_configFile, workload_folder, patching_config)
  File "/HiBench3/bin/functions/load_config.py", line 217, in load_config
    generate_optional_value()
  File "/HiBench3/bin/functions/load_config.py", line 641, in generate_optional_value
    probe_masters_slaves_hostnames()
  File "HiBench3/bin/functions/load_config.py", line 577, in probe_masters_slaves_hostnames
    probe_masters_slaves_by_Yarn()
  File "/HiBench3/bin/functions/load_config.py", line 528, in probe_masters_slaves_by_Yarn
    assert 0, "Get workers from yarn-site.xml page failed.
AssertionError: Get workers from yarn-site.xml page failed.
[17:50:32] Parsing conf: /home/team7/HiBench3/conf/hadoop.conf
[17:50:32] Parsing conf: /home/team7/HiBench3/conf/hibench.conf
[17:50:32] Parsing conf: /home/team7/HiBench3/conf/spark.conf
[17:50:32] Parsing conf: /home/team7/HiBench3/conf/workloads/micro/terasort.conf
[17:50:34] display configuration commit list
------------------------------------------------------------
No.        CommitId       Label                          User          TimeStamp
------------------------------------------------------------
1          1001225122     td-begin_backroll              root          2022-11-20 17:50:32
2          1001225121     -                              root          2022-11-20 17:50:31
------------------------------------------------------------
```

日志

```
2023-01-14 23:05:14 INFO:  Reading data from /user/input/file.txt
2023-01-14 23:05:14 DEBUG: Setting block size to 1919810
2023-01-14 23:05:14 INFO:  Setting replication factor to 4
2023-01-14 23:05:14 ERROR: /user/input/file.txt does not exist
```

结构化数据

Date	Time	Level	Template	Parameters
2023/1/14	23:05:14	INFO	Reading data from <*>	['/user/input/file.txt']
2023/1/14	23:05:14	DEBUG	Setting block size to <*>	['1919810']
2023/1/14	23:05:14	INFO	Setting replication factor to <*>	['4']
2023/1/14	23:05:14	ERROR	<*> does not exist	['/user/input/file.txt']

[1] ESEC/FSE'23 Hue: A User-Adaptive Parser for Hybrid Logs

传统日志分析的挑战：日志解析

3. 受限于方法本身的约束

设计者通常基于自身假设构建解析规则，然而日志消息并非总是服从某一特定规则。例如Drain [2] 做出了如下假设：

1. 属于同一事件的日志消息都具有相同的日志长度
2. 日志消息靠前的单词都更可能属于模版而非参数

这些假设在Proxifier [3]数据集上则几乎无法成立。因为Proxifier具备长度可变的模版，且参数都聚集在靠前位置

日志

```
2023-01-14 23:05:14 INFO:  Reading data from /user/input/file.txt
2023-01-14 23:05:14 DEBUG: Setting block size to 1919810
2023-01-14 23:05:14 INFO:  Setting replication factor to 4
2023-01-14 23:05:14 ERROR: /user/input/file.txt does not exist
```

结构化数据

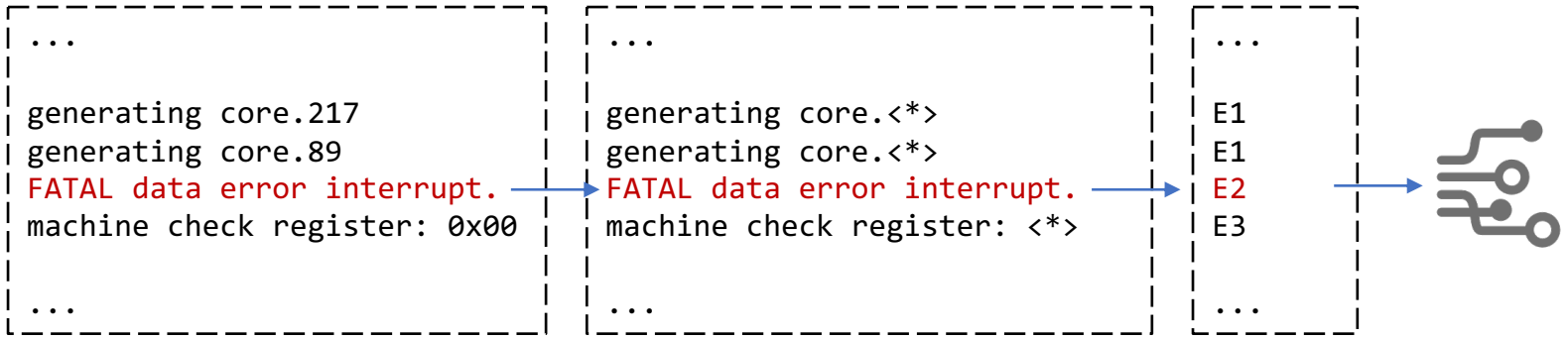
Date	Time	Level	Template	Parameters
2023/1/14	23:05:14	INFO	Reading data from <*>	['/user/input/file.txt']
2023/1/14	23:05:14	DEBUG	Setting block size to <*>	['1919810']
2023/1/14	23:05:14	INFO	Setting replication factor to <*>	['4']
2023/1/14	23:05:14	ERROR	<*> does not exist	['/user/input/file.txt']

[2] ICWS'17 Drain: An online log parsing approach with fixed depth tree
[3] ISSRE'23 Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics

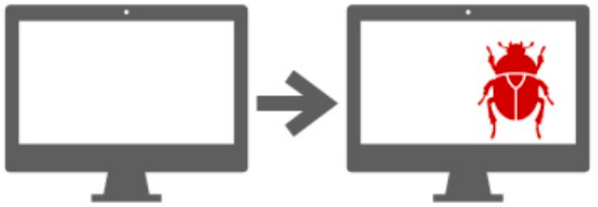
传统日志分析的挑战：日志挖掘

1. 难以挖掘语义信息

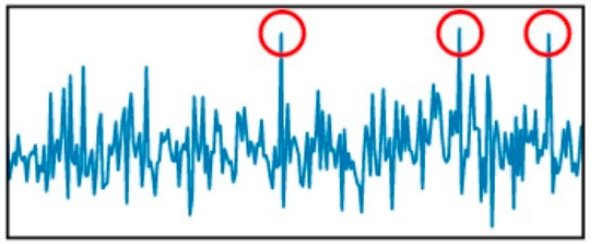
当前日志挖掘方案通常仅利用了简单的统计信息，而无法将日志的语义信息纳入其中。这导致消息文本中的大量信息都在挖掘的过程中被抛弃。



（语义信息丢失）



故障诊断



异常检测

传统日志分析的挑战：日志挖掘

2. 难以适应日志的迭代

在实际开发中，由于业务代码的日常迭代和服务的上线下线，日志的正常和异常模式是动态改变的[4]。

Case 1:

```
Creating and opening new * channel factory
```

```
Creating and opening new * channel factory for bindingId *
```

Case 2:

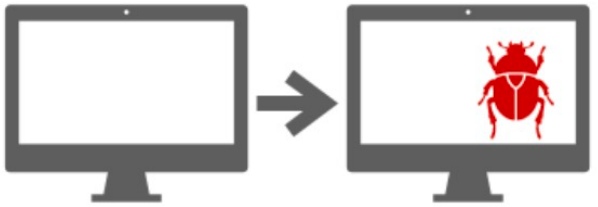
```
{"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
```

```
MiniMerge: {"Result":* "BaseHash":* UploadHash":* TipHash":* NewTipHash":* }
```

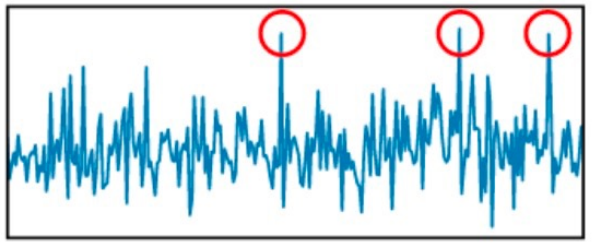
Case 3:

```
PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ] [IsADocumentSession: * ]] [IsClear: * ]
```

```
PutSharedRemoteServiceLocation - Entering with inputs [LocationType: * ]
[SrsLocation: [Server: * ] [Cluster: * ] [Session: * ] [BackupServer: * ] [Back-
upSession: * ] [Environment: * ]] [IsClear: * ]
```



故障诊断



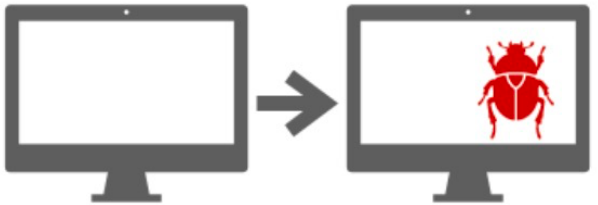
异常检测

[4] ESEC/FSE'19 Robust Log-Based Anomaly Detection on Unstable Log Data

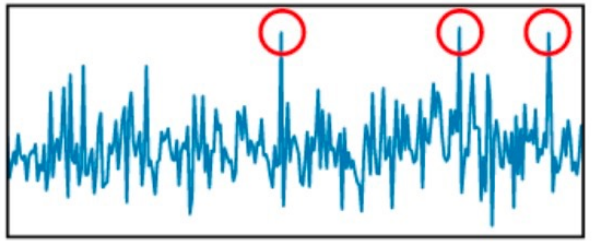
传统日志分析的挑战：日志挖掘

3. 受限于上游日志分析步骤

日志挖掘方法通常和上游的日志记录和日志解析方法相互耦合，因此也会受到方法设计上的强假设约束影响。



故障诊断

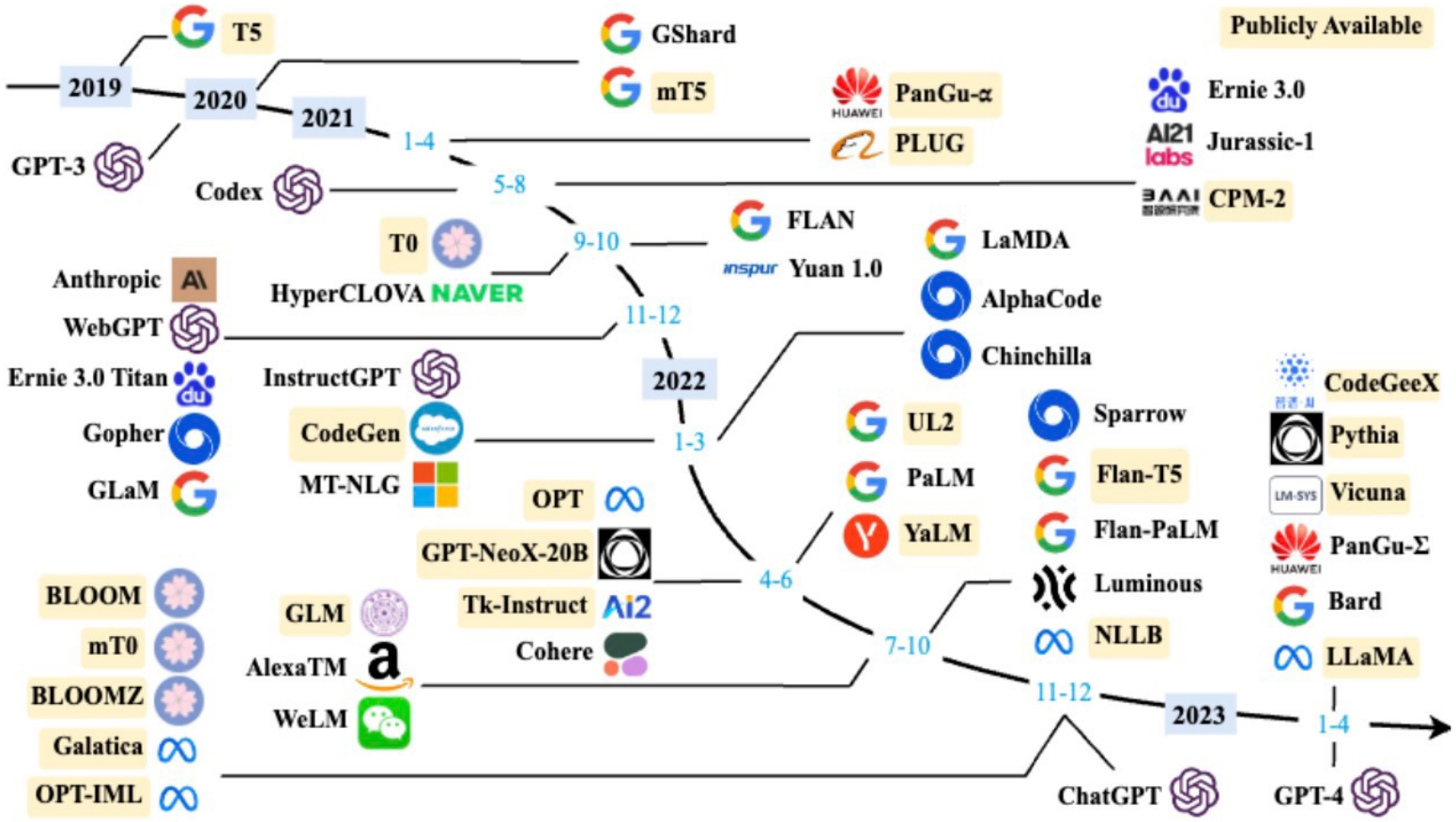


异常检测

PART 03

大模型驱动的日志分析：技术路线

日志分析的机遇：大语言模型



在大模型时代，日志无需进行额外的表征，即可直接作为大模型的输入，从而利用大模型的强大推理能力。

* 图摘取自 复旦大学《大规模语言模型：从理论到实践》

日志分析的机遇：大语言模型

站在巨人的肩膀上：预训练的大模型带来的便捷和高效

* 图摘取自 复旦大学《大规模语言模型：从理论到实践》

日志分析的机遇：大语言模型

模型名称	发布时间	模型参数量	基础模型	模型类型	预训练数据量
T5[19]	2019 年 10 月	110 亿	-	语言模型	1 万亿 Token
mT5[26]	2020 年 10 月	130 亿	-	语言模型	1 万亿 Token
PanGu-α[22]	2021 年 4 月	130 亿	-	语言模型	1.1 万亿 Token
CPM-2[27]	2021 年 6 月	1980 亿	-	语言模型	2.6 万亿 Token
T0[28]	2021 年 10 月	110 亿	T5	指令微调模型	–
CodeGen[29]	2022 年 3 月	160 亿	-	语言模型	5770 亿 Token
GPT-NeoX-20B[30]	2022 年 4 月	200 亿	-	语言模型	825GB 数据
OPT[31]	2022 年 5 月	1750 亿	-	语言模型	1800 亿 Token
GLM[32]	2022 年 10 月	1300 亿	-	语言模型	4000 亿 Token
Flan-T5[23]	2022 年 10 月	110 亿	T5	指令微调模型	-
BLOOM[33]	2022 年 11 月	1760 亿	-	语言模型	3660 亿 Token
Galactica[34]	2022 年 11 月	1200 亿	-	语言模型	1060 亿 Token
BLOOMZ[35]	2022 年 11 月	1760 亿	BLOOM	指令微调模型	-
OPT-IML[36]	2022 年 12 月	1750 亿	OPT	指令微调模型	-
LLaMA[37]	2023 年 2 月	652 亿	-	语言模型	1.4 万亿 Token
MOSS	2023 年 2 月	160 亿	Codegen	指令微调模型	-
ChatGLM-6B[32]	2023 年 4 月	62 亿	GLM	指令微调模型	-
Alpaca[38]	2023 年 4 月	130 亿	LLaMA	指令微调模型	-
Vicuna[39]	2023 年 4 月	130 亿	LLaMA	指令微调模型	-
Koala[40]	2023 年 4 月	130 亿	LLaMA	指令微调模型	-

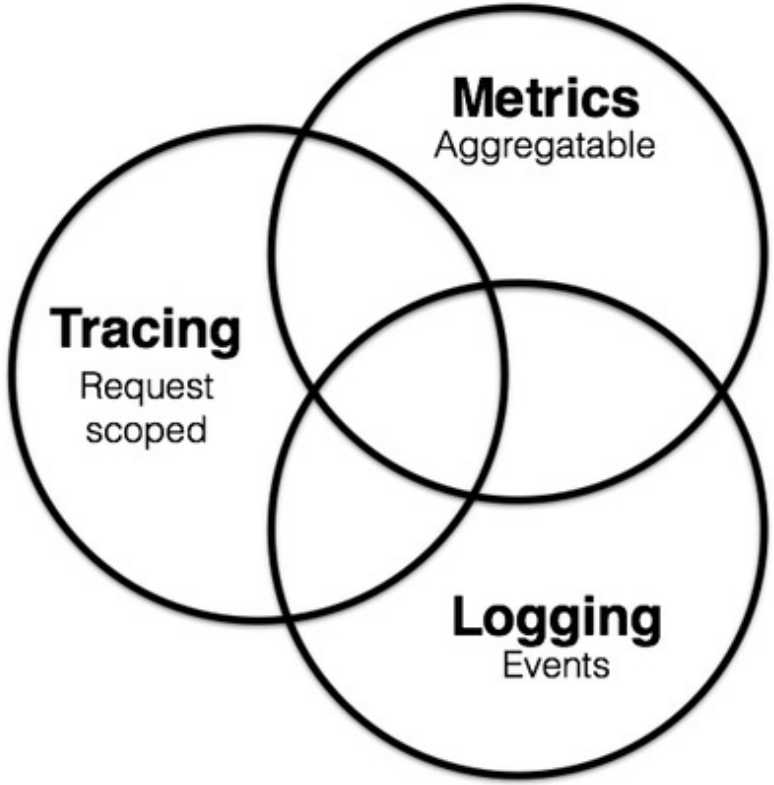
大量开源大模型能够支持定制化的垂类模型

* 图摘取自 复旦大学《大规模语言模型：从理论到实践》

日志分析的机遇：大语言模型

日志是一种文本模态的数据

和自然语言或程序语言类似，日志通常是不具备固定格式，且富含语义信息的文本数据。而运维领域另外两种重要的数据（调用链和指标）则不具备这种特性：调用链是结构数据，而指标通常是时序数据。这使得日志分析可能成为大模型时代运维领域发展最迅速的方向。



日志分析的机遇：大语言模型

日志分析仅涉及文本模态数据

无论是日志记录、日志解析、日志压缩、还是日志挖掘，其输入和输出都是文本数据（代码或日志）。这使得日志分析可能成为大模型时代运维领域发展最迅速的方向。

```java
public void setTemperature(Integer temperature) {
  // ...
  logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);
  if (temperature.intValue() > 50) {
    logger.info("Temperature has risen above 50 degrees.");
  }
}
```

⇓

```
0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.
0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

PART 04

大模型驱动的日志分析：近期工作

近期工作：以日志记录为例

日志记录

即，在业务代码中插入合适的日志语句。

日志记录的目标如下：
1. 保证故障信息或关键事件能够被记录在日志中
2. 保证记录的日志能够用于甄别不同的事件或故障
3. 标准化日志规范，减少冗余日志输出造成的开销

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     client.flush();
5|     client.close().addListener(CLOSE);
6|}
```

细分目标#1：在合适的位置处插入日志语句：Line#4

细分目标#2：设置合适的日志等级：LOG.Info()

细分目标#3：记录合适的日志消息："Disconnect successful, clientID: {}"

```
1| public void processDisconnect(Channel client) {
2|     String clientID = NettyAttrManager.getAttrClientId(client);
3|     cleanWillMessage(clientID);
4|     LOG.info("Disconnect successful, clientID: {}", clientID);
5|     client.flush();
6|     client.close().addListener(CLOSE);
7|}
```

近期工作：日志记录技术UniLog

```
<line0> public void processDisconnect(Channel client) {
<line1>     String clientID = NettyAttrManager.getAttrClientId(client);
<line2>     cleanWillMessage(clientID);
<line3>     client.flush();
<line4>     client.close().addListener(CLOSE);
<line5> }
```

传统方法

UniLog

子任务#1：在合适的位置处插入日志语句：Line#4
子任务#2：设置合适的日志等级：LOG.Info()
子任务#3：记录合适的日志消息："Disconnect successful, clientID: {}"

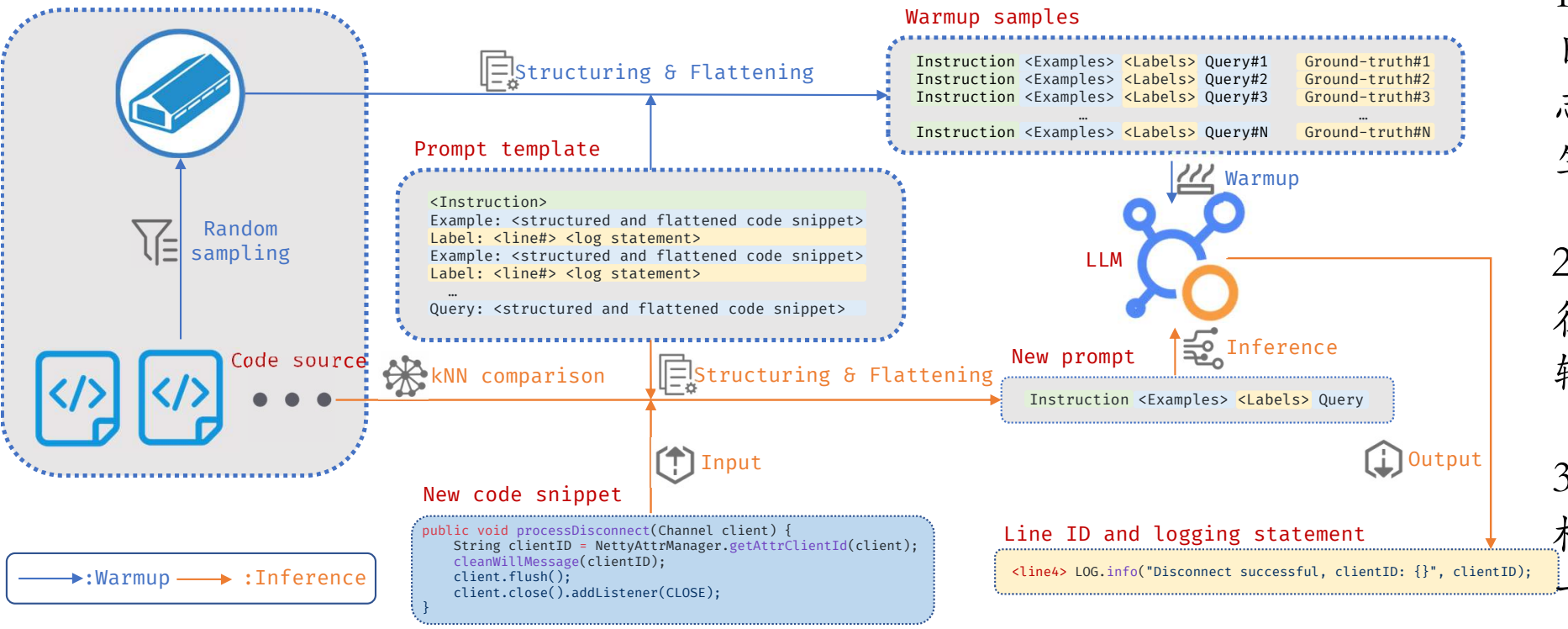
```
<line3> LOG.info(\"Disconnect successful, clientID: {}\", clientID);
```

UniLog [5]:

UniLog是第一个基于上下文学习 (in-context learning, ICL) 的日志记录框架，提供了新的日志记录范式：将所有日志记录的细分任务统一构建成一个文本生成任务，端到端地实现日志记录。

[5] ICSE'24 UniLog: Automatic Logging via LLM and In-Context Learning

近期工作：日志记录技术UniLog



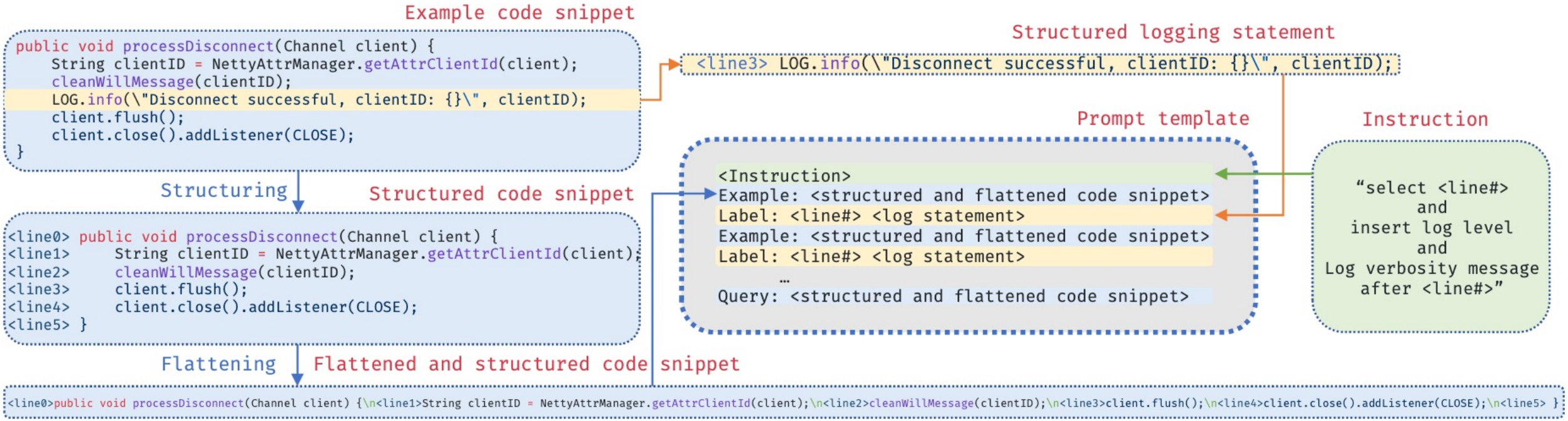
1. UniLog使用行号机制，将日志打印位置的预测任务和日志语句的生成任务统合为文本生成任务

2. UniLog针对每个Query的特征，自动化构建最优Prompt输入模型

3. UniLog通过随机抽样的ICL样本对模型进行预热，可以进一步提升模型的日志记录精度。

[5] ICSE'24 UniLog: Automatic Logging via LLM and In-Context Learning

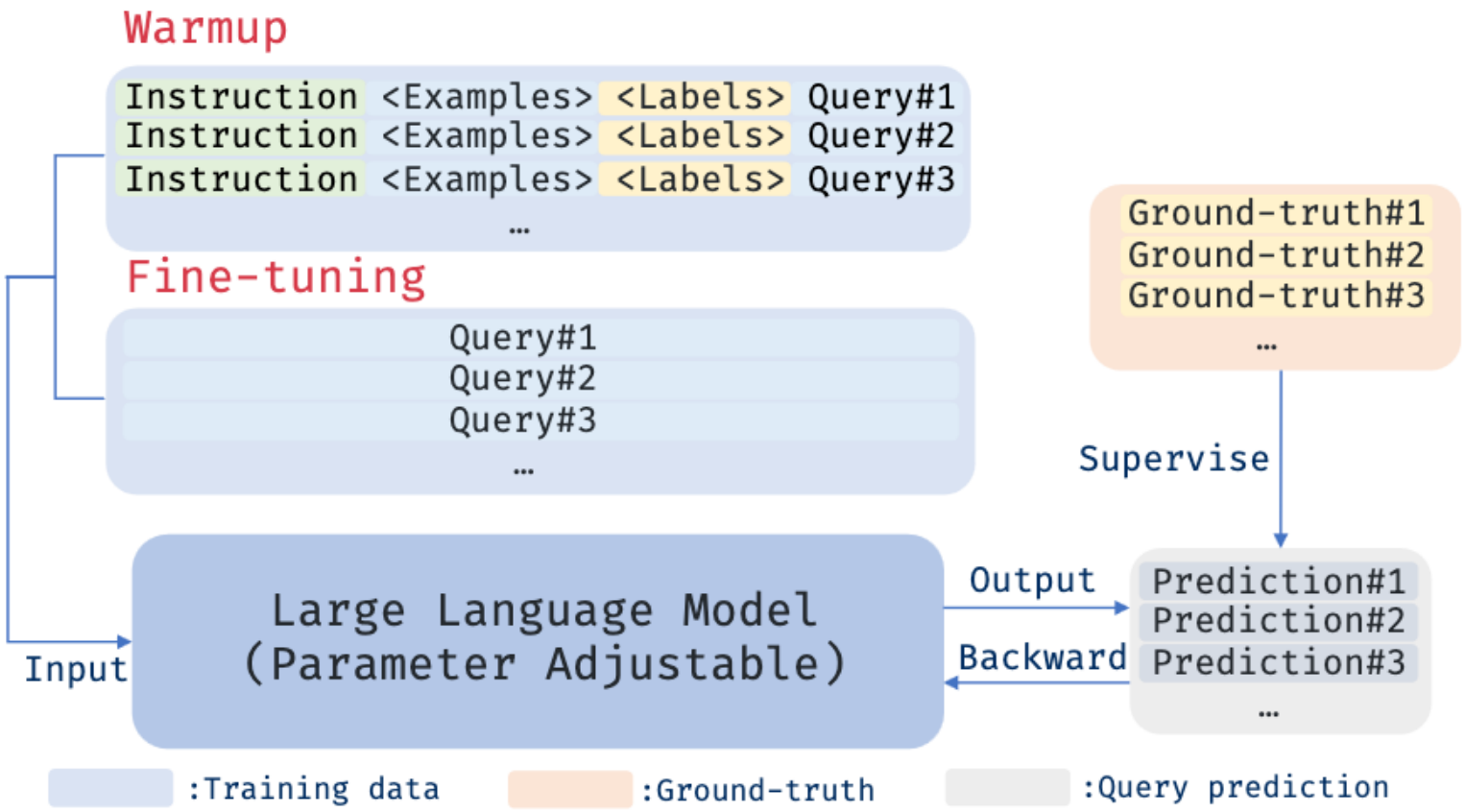
近期工作：日志记录技术UniLog



我们设计了一套自动化流水线来为每个目标代码构建提示输入例和参考输出结果。

[5] ICSE’24 UniLog: Automatic Logging via LLM and In-Context Learning

近期工作：日志记录技术UniLog



采用极少量的随机样本对模型进行预热可以进一步提升日志记录性能

实际实验证明预热样本仅需微调完整模型的4%的数据，即可做到比微调效果更好的日志记录表现。

[5] ICSE'24 UniLog: Automatic Logging via LLM and In-Context Learning

近期工作：UniLog效果评估

Method	PA	LA	MA	CLA	CMA	BLEU
LANCE-Full	60.2	60.4	13.7	73.2	21.4	N/A
LANCE-Best	65.4	66.2	16.9	76.8	24.3	N/A
UniLog-w/o warmup	66.5	66.8	20.2	75.7	28.7	24.5
UniLog-w/ warmup	76.9	72.3	22.4	77.9	28.1	27.1

UniLog在日志位置、日志等级、日志消息的精度上都远超当前基于机器翻译范式的Baselines。

[5] ICSE'24 UniLog: Automatic Logging via LLM and In-Context Learning

近期工作：UniLog效果评估

```
public class A {
  protected List<Blob> loadBlobs(List<Map<String, String>> blobInfos) {
    log.debug("Loading blobs from the file system: " + blobInfos);
    List<Blob> blobs = new ArrayList<>();
    for (Map<String, String> info : blobInfos) {
      File blobFile = new File(cacheDir, info.get("file"));
      Blob blob = new FileBlob(blobFile);
      blob.setEncoding(info.get("encoding"));
      blob.setMimeType(info.get("mimetype"));
      blob.setFilename(info.get("filename"));
      blob.setDigest(info.get("digest"));
      blobs.add(blob);
    }
    log.debug("Loaded blobs: " + blobs);
    return blobs;
  }
}
```

: Prediction　　:Ground-truth

有时UniLog会在不同的位置生成不同的日志语句。这些日志语句完全不一样，导致在自动化评估的时候被判定为错例。然而，我们发现部分错例在实际工程中仍然是有意义的。

[5] ICSE'24 UniLog: Automatic Logging via LLM and In-Context Learning

近期工作：以日志解析为例

日志解析

即，提取日志模版，将半结构日志解析为结构数据。

日志解析的目标如下：
1. 提取日志的公共部分并提交给日志压缩使用。
2. 将日志转变为结构化数据并提交给日志挖掘使用。

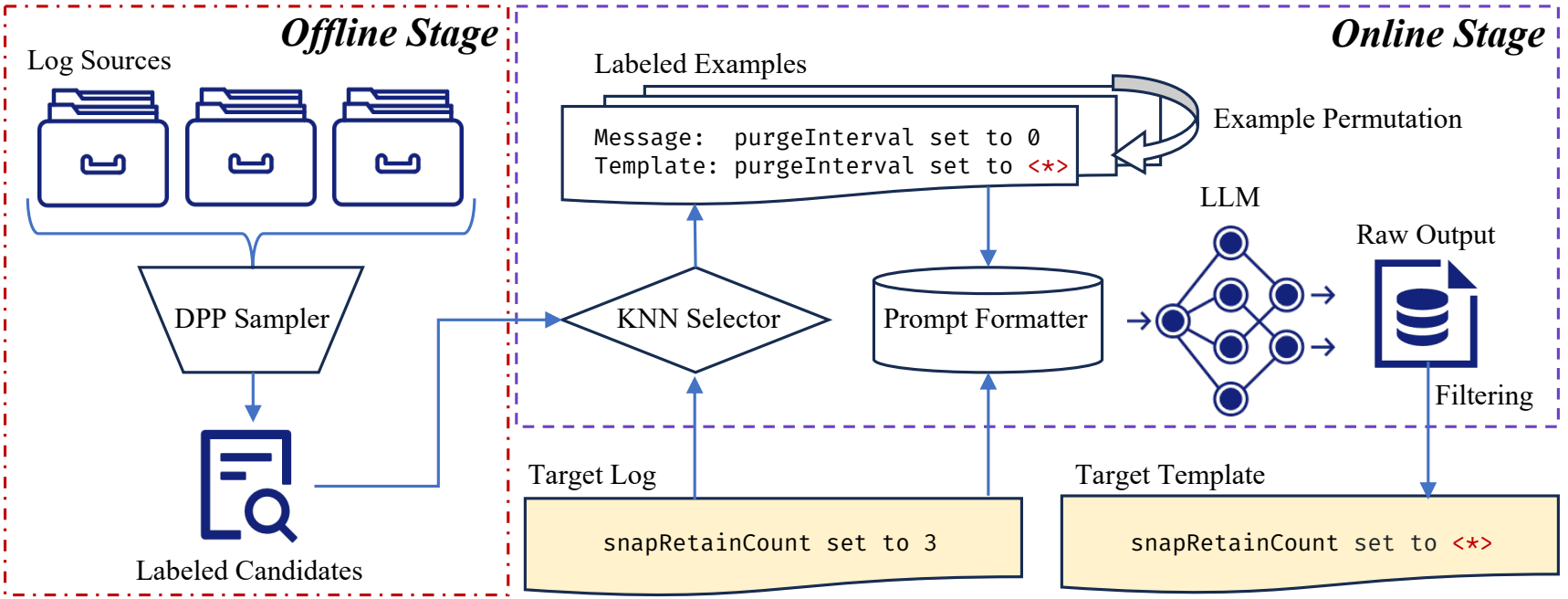
日志

```
2023-01-14 23:05:14 INFO:  Reading data from /user/input/file.txt
2023-01-14 23:05:14 DEBUG: Setting block size to 1919810
2023-01-14 23:05:14 INFO:  Setting replication factor to 4
2023-01-14 23:05:14 ERROR: /user/input/file.txt does not exist
```

结构化数据

Date	Time	Level	Template	Parameters
2023/1/14	23:05:14	INFO	Reading data from <*>	['/user/input/file.txt']
2023/1/14	23:05:14	DEBUG	Setting block size to <*>	['1919810']
2023/1/14	23:05:14	INFO	Setting replication factor to <*>	['4']
2023/1/14	23:05:14	ERROR	<*> does not exist	['/user/input/file.txt']

近期工作：日志解析技术DivLog



DivLog [6]:

DivLog是第一个基于上下文学习(in-context learning, ICL)的日志解析框架。通过主动增加样本多样性和目标相似度的方式，在不涉及梯度计算的条件下实现超高精度的有监督日志解析。

[6] ICSE'24 DivLog: Log Parsing with Prompt Enhanced In-Context Learning

近期工作：日志解析技术DivLog

Prompt

```
Extract one log template, \
substitute variable tokens in the log as <*> between <START> and <END> tags.


Message: Recalculating schedule, headroom=<memory:0, vCores:-27>
TEmplate: <START> Recalculating schedule, headroom=<memory:<*>, vCores:<*>> <END>

Message: Progress of TaskAttempt attempt_000001_0 is : 0.052178
Template: <START> Progress of TaskAttempt attempt_<*> is : <*> <END>

Message: task_000002 Task Transitioned from SCHEDULED to RUNNING
Template: <START> task_<*> Task Transitioned from SCHEDULED to RUNNING <END>

Message: attempt_000001 TaskAttempt Transitioned from ASSIGNED to RUNNING
Template: <START> attempt_<*> TaskAttempt Transitioned from ASSIGNED to RUNNING <END>

Message: JVM with ID : jvm_000003 asked for a task
Template: <START> JVM with ID : jvm_<*> asked for a task <END>


Message: JVM with ID: jvm_000005 given task: attempt_000003_0
```

Raw Output

```
<START> JVM with ID: jvm_<*> given task: attempt_<*> <END>
          ...
          ...
(Other Redundant Text Contents)
```

Filtered Output

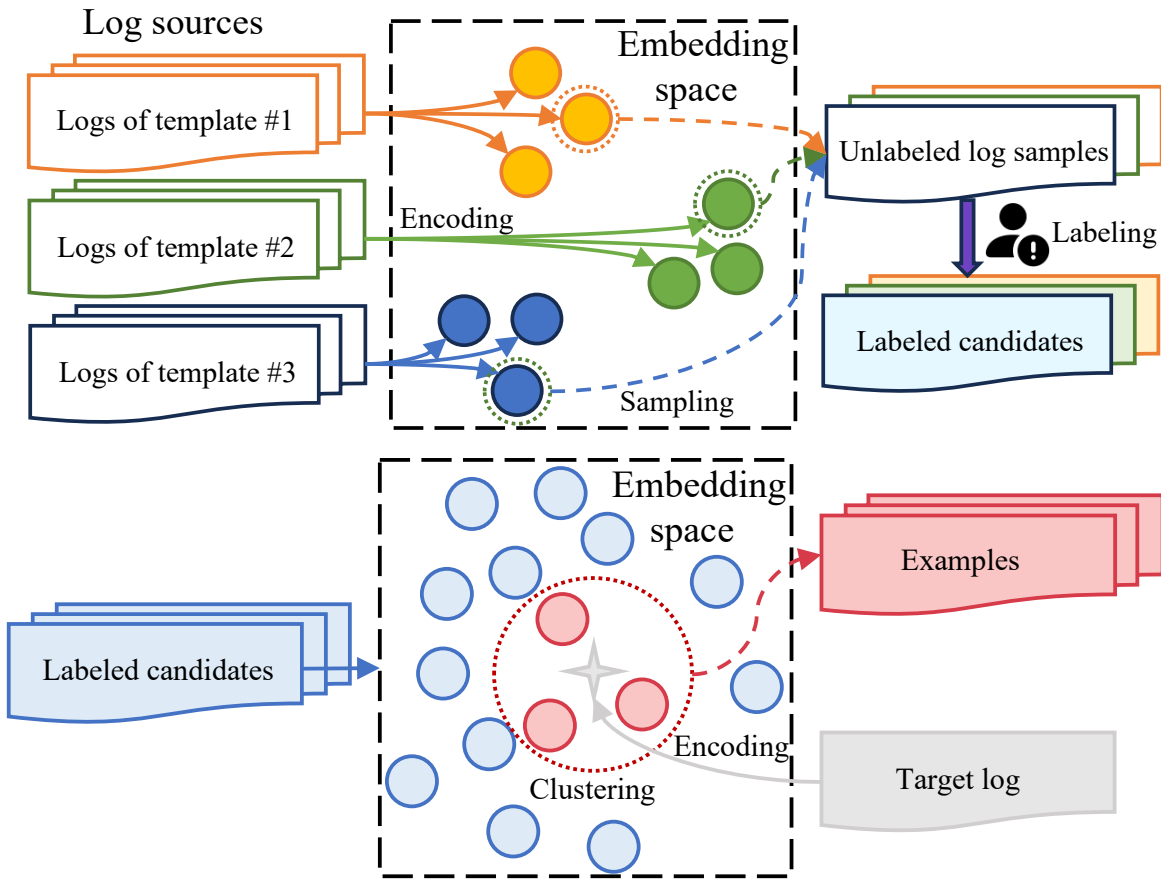
```
JVM with ID: jvm_<*> given task: attempt_<*>
```

DivLog [6]:

DivLog是第一个基于上下文学习(in-context learning, ICL)的日志解析框架。通过主动增加样本多样性和目标相似度的方式，在不涉及梯度计算的条件下实现超高精度的有监督日志解析。

[6] ICSE'24 DivLog: Log Parsing with Prompt Enhanced In-Context Learning

近期工作：日志解析技术DivLog



1. DivLog 通过极大化样本多样性的方式，提取一批无标签日志样本交付专家并确定其样本标签，以提高样本质量、减少有监督打标的负担

2. DivLog基于目标日志的向量余弦相似度，检索最相似的少量日志样本作为解析参考样本

3. DivLog利用特殊的定位符以从大模型的回复中精准提取日志模版，避免了无效信息的干扰

[6] ICSE'24 DivLog: Log Parsing with Prompt Enhanced In-Context Learning

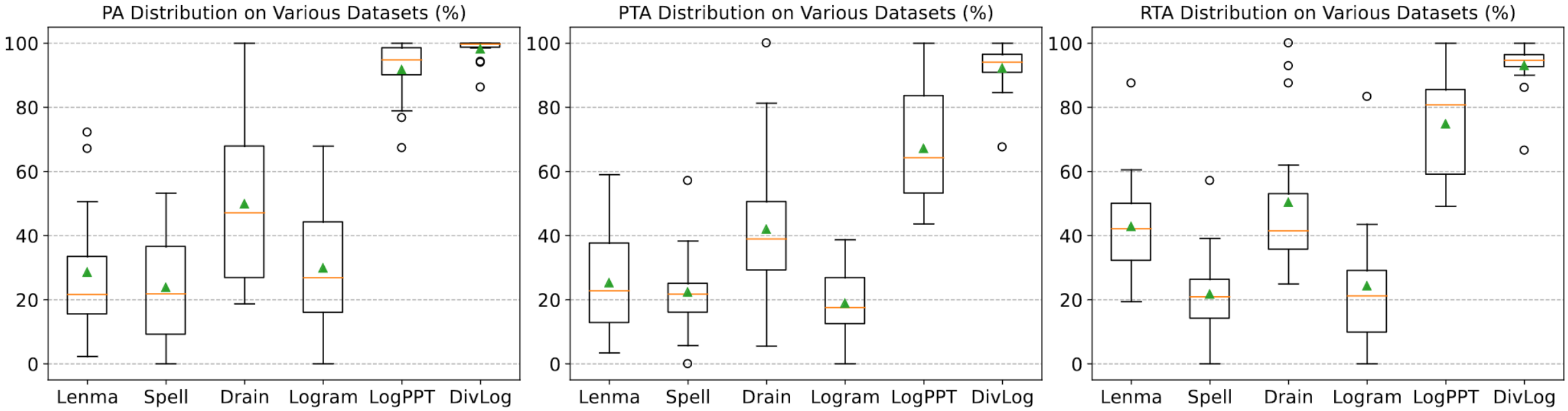
近期工作：DivLog效果评估

Dataset	LenMa			Spell			Drain			Logram			LogPPT			DivLog		
	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA	PA	PTA	RTA
Andriod	72.2	59.0	60.1	24.1	24.4	27.2	73.0	56.6	62.0	42.8	38.7	27.2	76.7	58.4	68.4	**94.3**	**84.6**	**86.1**
Apache	29.3	42.9	50.0	28.5	25.0	16.7	**100.0**	**100.0**	**100.0**	50.9	33.3	83.3	99.4	83.3	83.3	**100.0**	**100.0**	**100.0**
BGL	15.4	9.6	25.8	32.9	12.1	12.5	44.4	33.9	30.8	17.0	15.7	16.7	97.0	68.6	78.3	**98.7**	**91.7**	**92.5**
Hadoop	24.2	20.6	28.9	19.6	20.8	23.7	43.9	36.8	34.2	37.0	18.7	17.5	89.5	54.0	58.8	**99.6**	**98.3**	**99.1**
HDFS	12.5	37.5	42.9	48.7	57.1	57.1	95.9	81.3	92.9	1.8	19.4	42.9	90.2	85.7	85.7	**99.9**	**86.7**	**92.9**
HealthApp	28.9	3.4	46.7	15.2	21.5	18.7	24.1	8.3	34.7	26.3	3.5	25.3	78.9	85.3	85.3	**99.6**	**94.7**	**96.0**
HPC	67.1	12.9	50.0	53.2	38.3	39.1	67.2	38.8	41.3	67.9	37.7	43.5	94.7	73.6	84.8	**99.7**	**91.5**	**93.5**
Linux	13.2	41.0	41.4	10.9	17.3	14.7	19.4	43.4	42.2	18.5	13.5	4.3	94.9	47.5	49.1	**99.7**	**95.8**	**95.8**
Mac	15.5	14.6	19.4	3.3	5.7	5.9	27.7	21.2	24.9	25.2	16.4	20.2	67.3	43.6	53.4	**86.3**	**67.6**	**66.6**
OpenSSH	15.5	25.0	23.1	12.7	25.0	23.1	53.4	52.0	50.0	48.2	29.0	34.6	**97.6**	48.9	84.6	93.9	**96.2**	**92.6**
OpenStack	19.1	18.7	60.5	0.0	0.0	0.0	18.7	5.5	39.5	11.2	9.3	9.3	90.7	84.4	88.4	**99.9**	**95.3**	**95.3**
Proxifier	50.6	10.6	87.5	47.8	22.2	25.0	52.7	26.9	87.5	0.0	0.0	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Spark	2.3	12.4	33.3	33.6	34.5	27.8	37.6	50.0	41.7	27.5	3.1	22.2	99.1	60.0	58.3	**99.9**	**97.2**	**97.2**
Thunderbird	17.1	27.1	34.2	3.9	19.1	16.8	19.1	29.9	36.9	12.8	20.8	6.7	92.6	50.6	59.1	**98.5**	**91.8**	**95.3**
Windows	26.6	37.9	44.0	0.4	11.5	12.0	69.6	46.3	50.0	37.4	15.2	10.0	98.3	55.4	72.0	**99.8**	**91.8**	**90.0**
Zookeeper	45.7	30.0	36.0	45.3	22.0	26.0	49.8	39.1	36.0	51.6	26.1	24.0	99.0	74.1	86.0	**99.8**	**88.7**	**94.0**
Average	28.5	25.2	42.7	23.8	22.3	21.6	49.8	41.9	50.3	29.8	18.8	24.2	91.6	67.4	74.1	**98.1**	**92.1**	**92.9**

DivLog在多个日志数据集的消息和模版精度上都远超当前
所有的有监督和无监督的Baselines，几乎接近完美解析表现。

[6] ICSE'24 DivLog: Log Parsing with Prompt Enhanced In-Context Learning

近期工作：DivLog效果评估



DivLog在多个日志数据集上的解析表现浮动十分稳定，
表现出较高的可靠性

[6] ICSE'24 DivLog: Log Parsing with Prompt Enhanced In-Context Learning

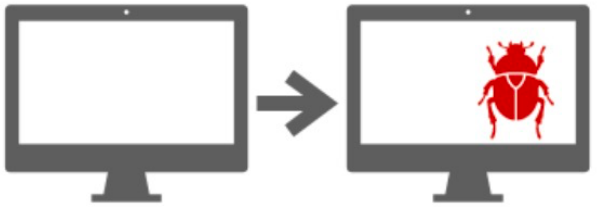
近期工作：以日志挖掘为例

故障诊断

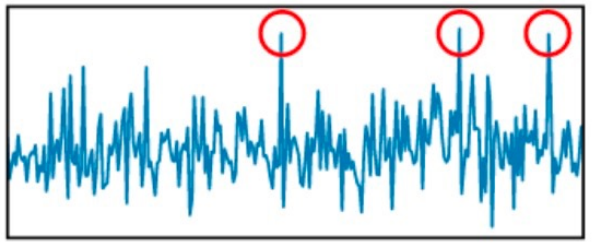
日志挖掘

即，将日志利用于多种保障软件可靠性的下游任务，包括：

1. 在故障发生时，进行故障诊断和定位
2. 在故障发生前，对故障模式进行预测
3. 系统异常状态的识别和检测

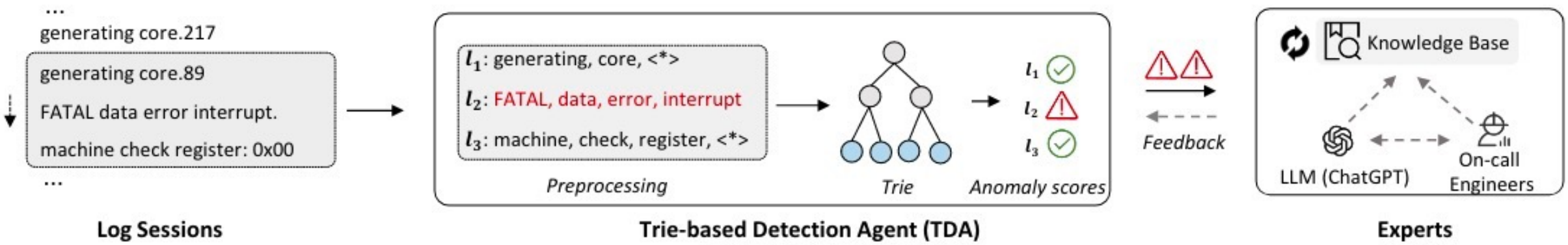


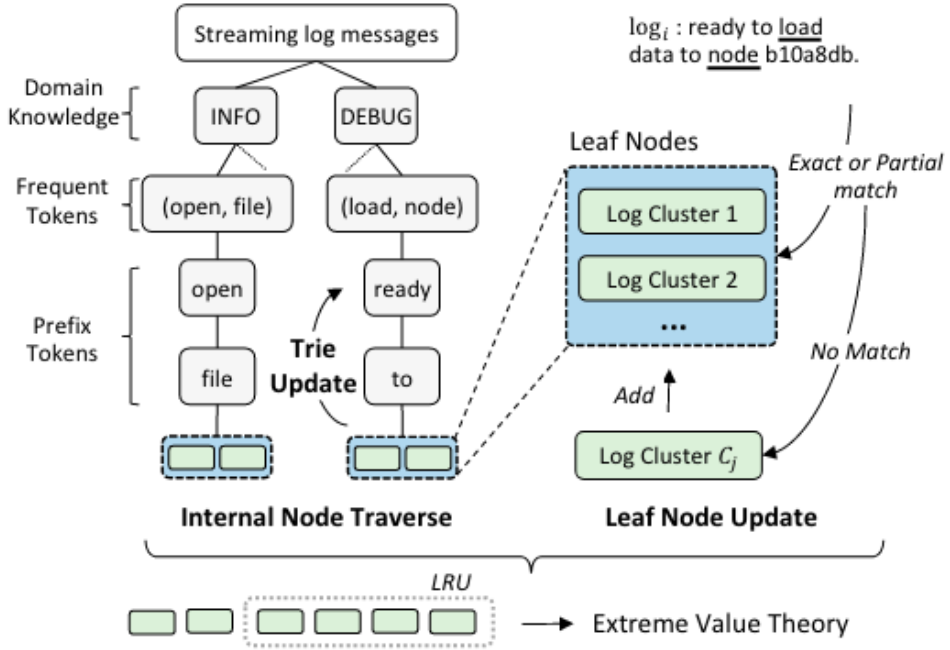
故障预测



异常检测

近期工作：以日志挖掘为例





ScaleAD [7]:

将大模型作为专家，对传统日志异常检测框架的结果进行评估，并对检测结果提供反馈，以帮助运维工程师缓解工作负担。

[7] arXiv Log-based Anomaly Detection based on EVT Theory with feedback

近期工作：以日志挖掘为例

Interpretable Log Parsing

Timestamp	*Template*	*Variable*	*Explanation*
1117842440	...	63543	Object amount: the number of instances of the "double-hummer alignment exceptions" that occurred.
1117842974		162	
1117843015		141	
1117848119	...	2	Object name: the object name that is referenced in the log.
		0x0b85eee0	Location indicator: the memory address of the object.
		0x05	Other parameters: additional parameters or flags associated with the object.
1117869872	...	172.18.24.3	Location indicator: the network location or address of the object.
1117869876		172.16.96.1	

↓

Interpretable Anomaly Detection

Is Anomaly?	*Explanation*
Yes.	The high number of "double-hummer alignment exceptions" and the failed attempts to read messages from a control stream suggest underlying issues with the system.

LogPrompt [8]:

将大模型作为专家，对每一条日志的内容进行解读。在解读过程中，LogPrompt要求大模型将思考的过程显式输出，以贴近人类运维工程师的方式进行日志异常检测。

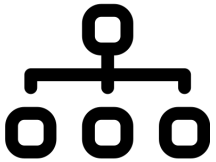
[8] arXiv LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis

PART 05

大模型驱动的日志分析：总结与展望

基于大模型的日志分析：总结


日志记录


日志解析

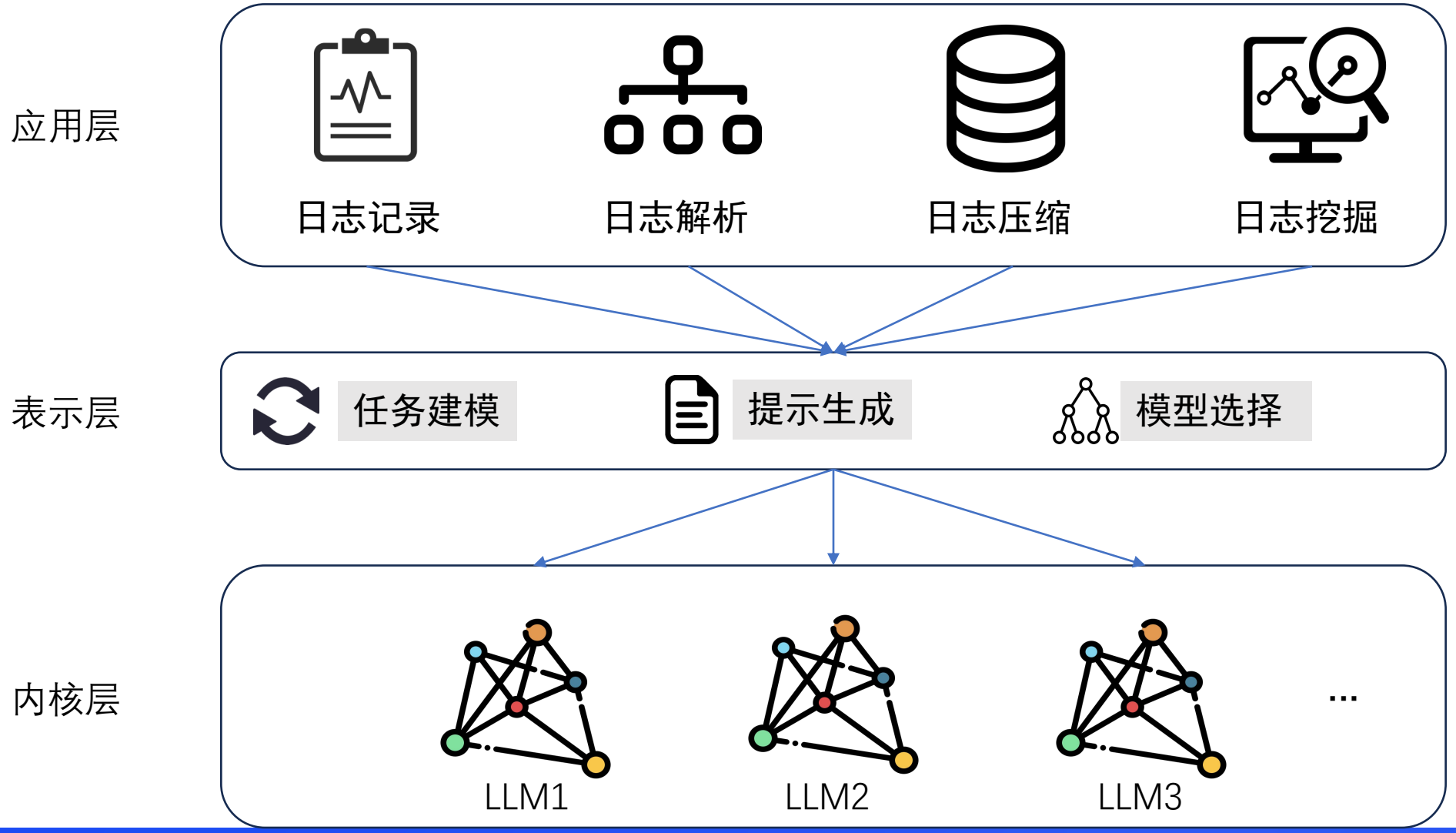

日志压缩


日志挖掘

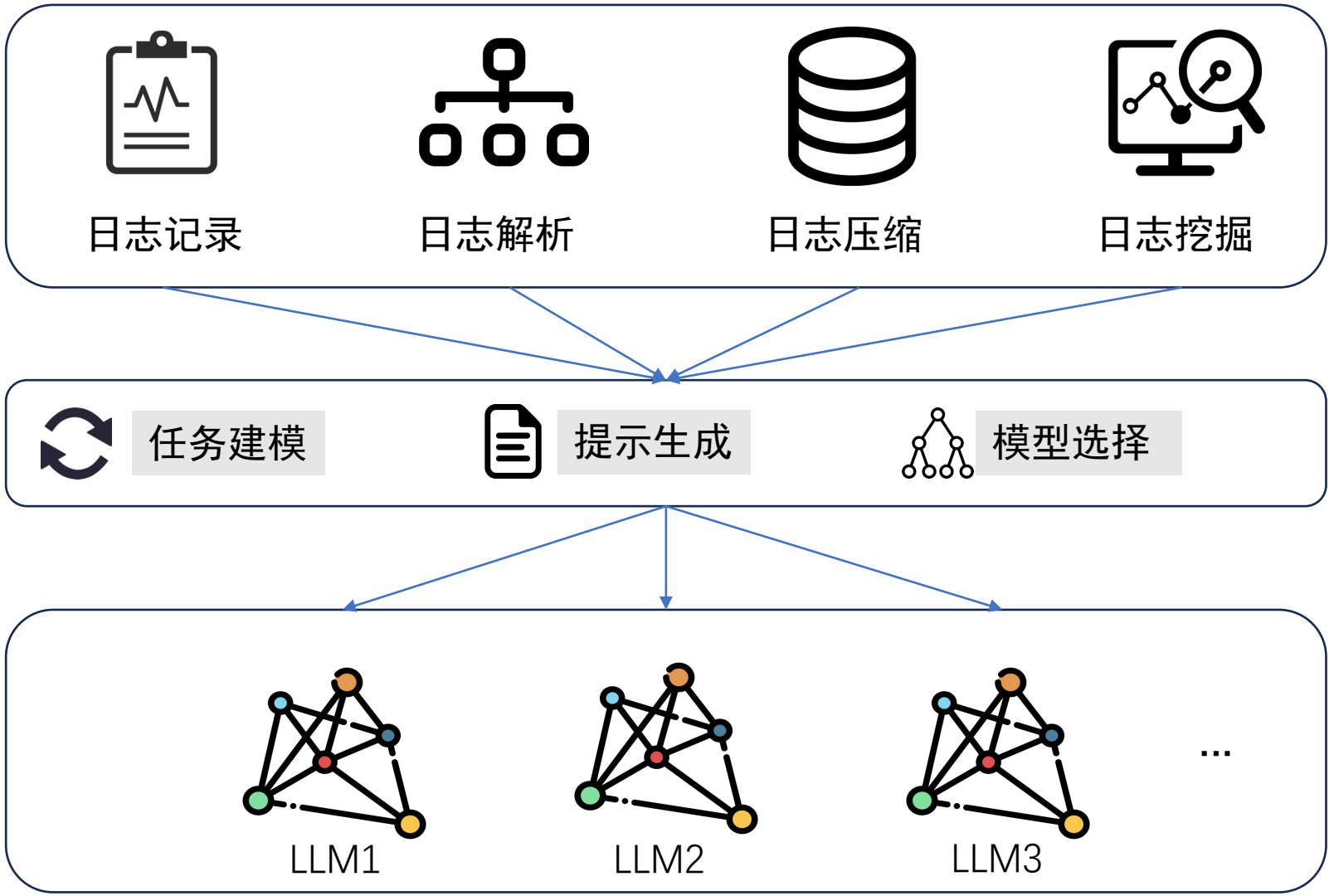
当前大模型驱动的日志分析趋势：

1. 从解决单一任务的范式逐渐转变为系统化整体化解决所有日志分析任务的范式
2. 从具体算法驱动的范式逐渐转变为高质量数据驱动的范式
3. 从机械地使用统计信息进行分析的范式逐渐转变为高效利用日志语义进行分析的范式

基于大模型的日志分析：展望



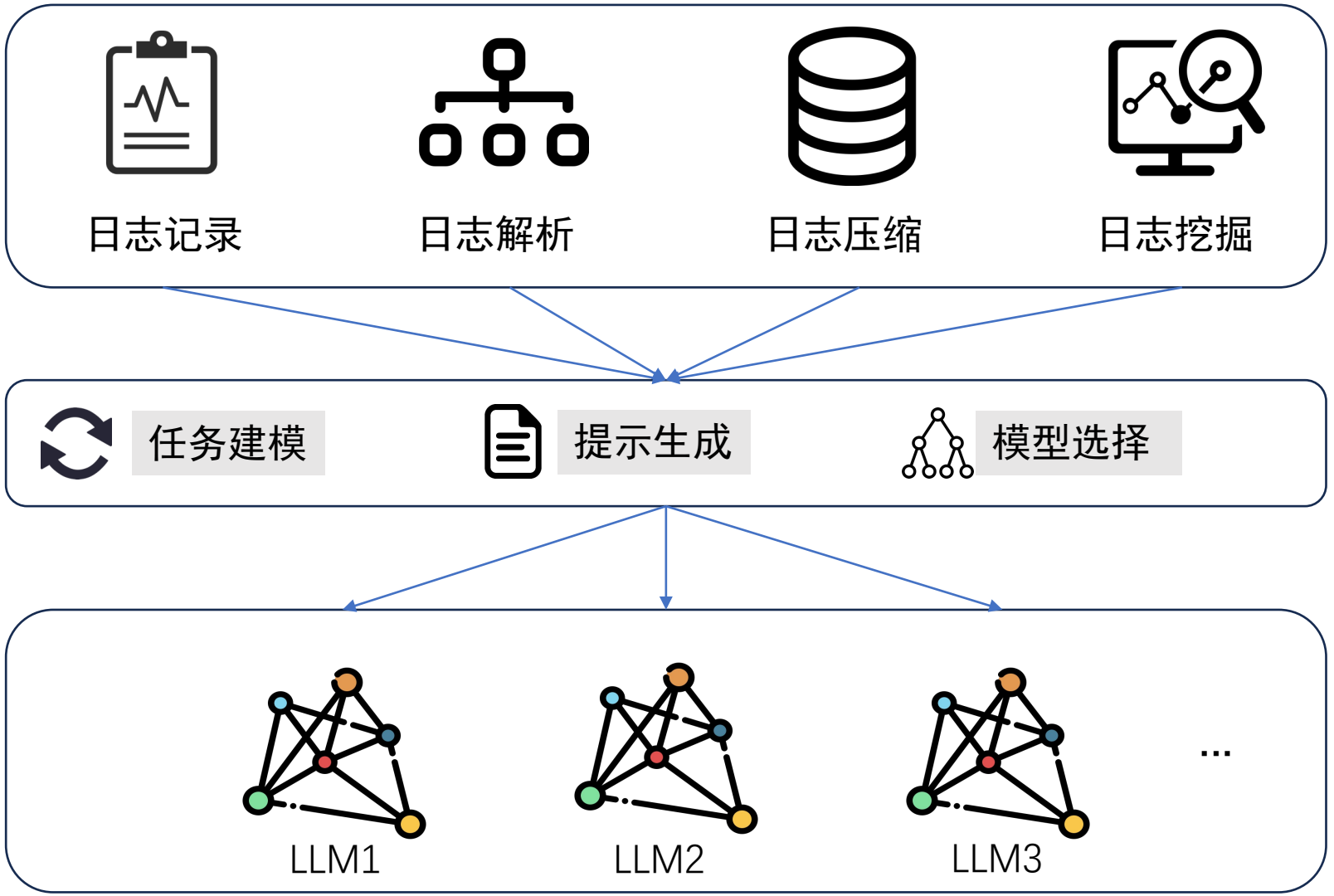
基于大模型的日志分析：展望



大模型驱动日志分析的未来：

1. 探索基于大模型驱动的日志压缩
2. 将所有日志分析任务整合为一个统一的大模型驱动的任务
3. 构建大模型驱动的日志分析平台，一站式解决不同复杂场景下的需要

基于大模型的日志分析：展望



任务建模：

将具体的日志分析任务，根据其自身特征，建模成可以用大模型解决的文本到文本的生成任务。

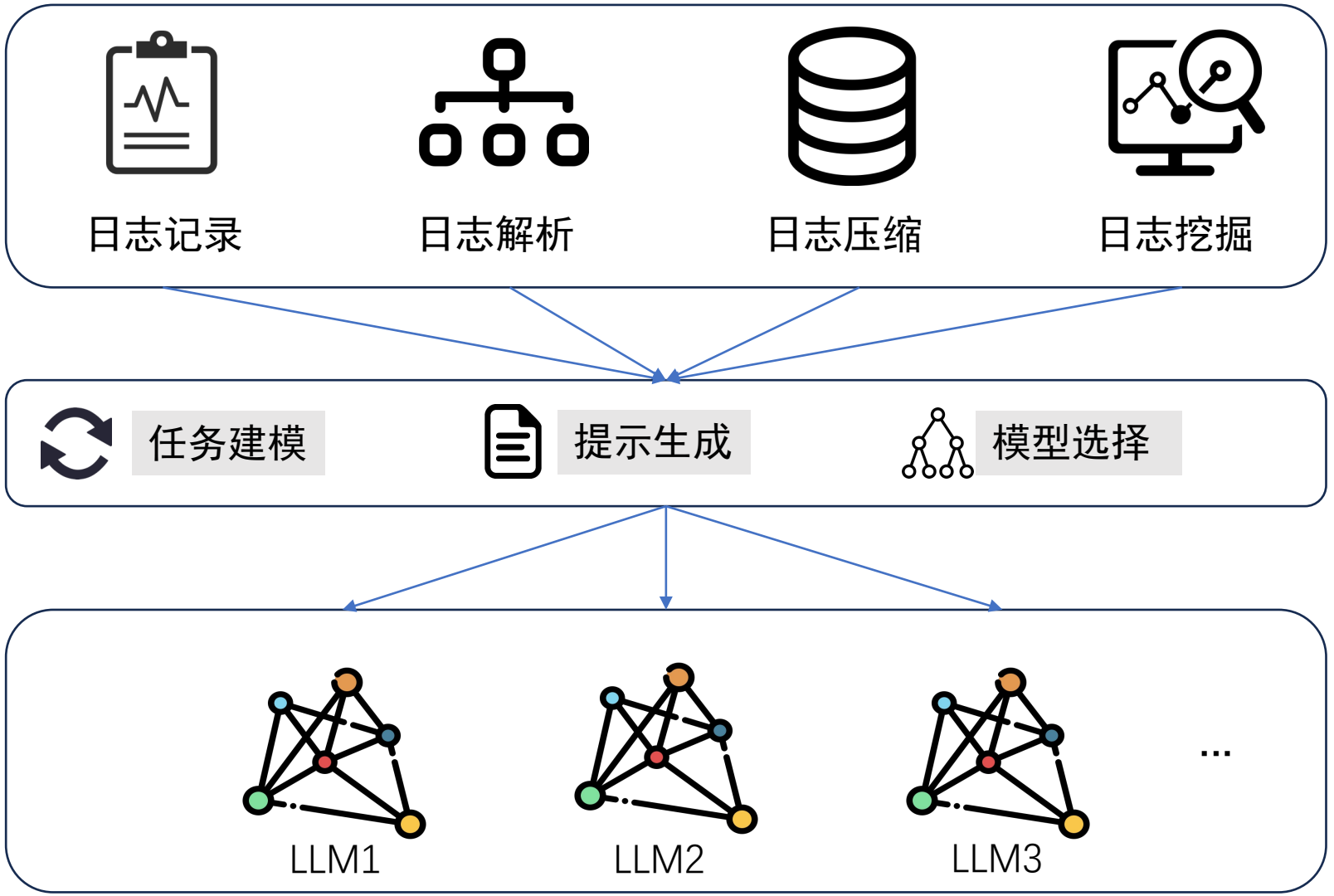
例如：

- 将日志记录从混合多个复杂子目标的任务，建模成日志埋点位置和日志语句的生成任务。

- 将日志解析从日志模版提取的词性分类任务，建模为日志模版生成任务。

- 将各类日志挖掘任务转化成各类生成、翻译、总结、或摘要等任务。

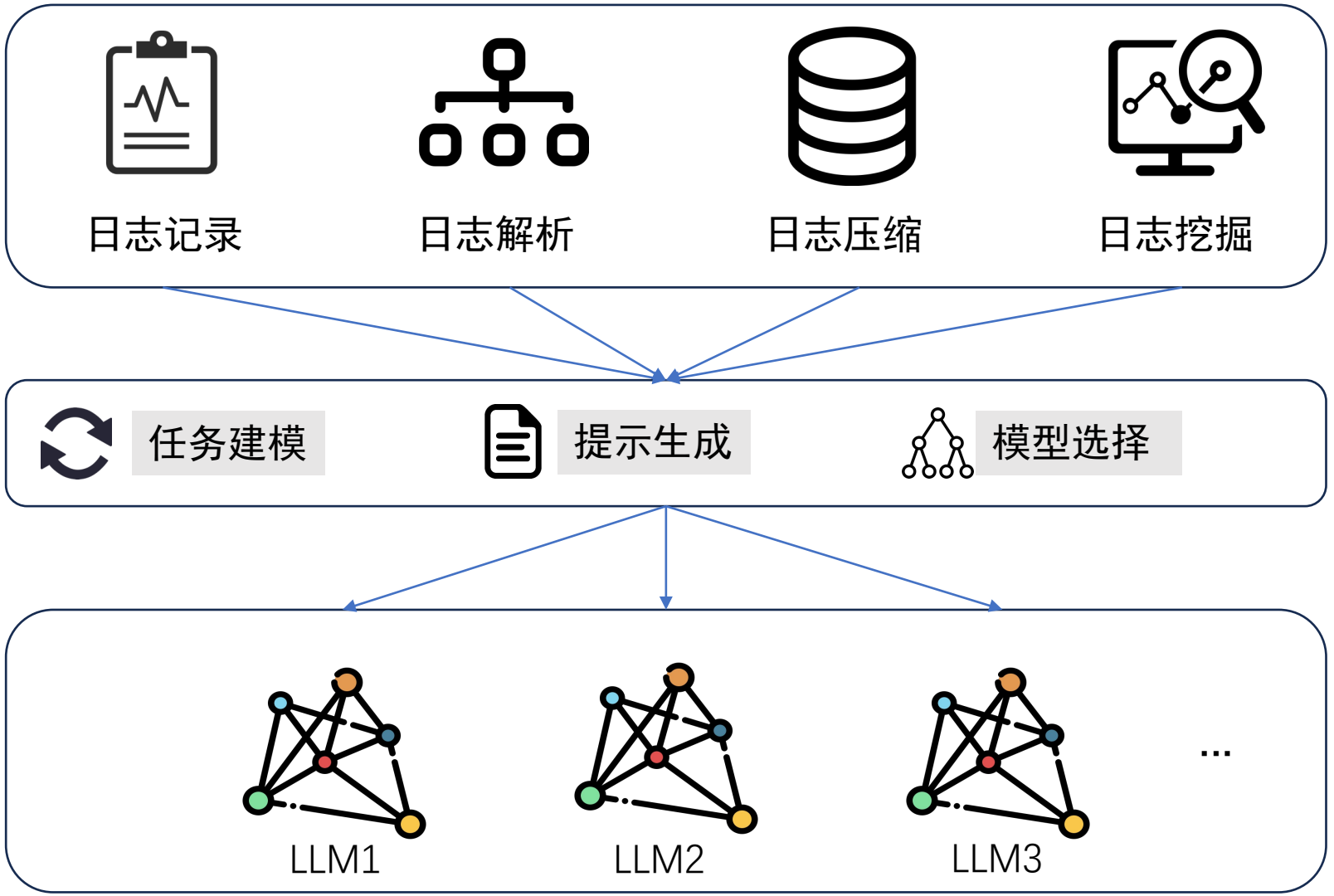
基于大模型的日志分析：展望



提示生成：

针对不同的任务建模，提供不同的提示词。其中提示词可能包含硬提示（即离散的文本提示词），或软提示（即连续的向量或分布）。提示词将促进大模型理解具体的日志分析任务

基于大模型的日志分析：展望



模型选择：

日志分析框架中的大模型内核可以同时包括有：

1. 通用基础模型
2. 通用指令微调模型
3. 针对日志微调的模型
4. 针对具体日志分析任务微调的模型

应当根据任务的特性选择模型，例如：
- 根据任务难度，选择不同体量的模型
- 根据任务是否涉及交叉知识，选择垂类模型或通用模型

AiDD AI+ 软件研发数字峰会
AI+ software Development Digital summit

THANKS