Large Language Models for Anomaly Detection in Computational Workflows: From Supervised Fine-Tuning to In-Context Learning

Hongwei Jin, George Papadimitriou, Krishnan Raghavan, Pawel Zuk, Prasanna Balaprakash, Cong Wang, Anirban Mandal, Ewa Deelman

PRESENTER:

SUSHIL RAJ REGMI

UNIVERSITY OF NORTH TEXAS

Anomaly Detection in computational workflows

- Anomalies arise due to high usage of computing resources, memory consumption and I/O operations

Traditional methods for anomaly detection:

  - Rule based systems (Threshold-Based detection, Heuristics, Conditional logic)
  - Statistical Analysis (Z-Score Analysis, IQR, Moving Average, Chi-Square Test)
  - Machine learning (Autoencoders, Isolation Forest, PCA, Random Forest, LSTM)

Anomaly Detection in computational workflows

Limitations of Traditional Anomaly Detection Methods

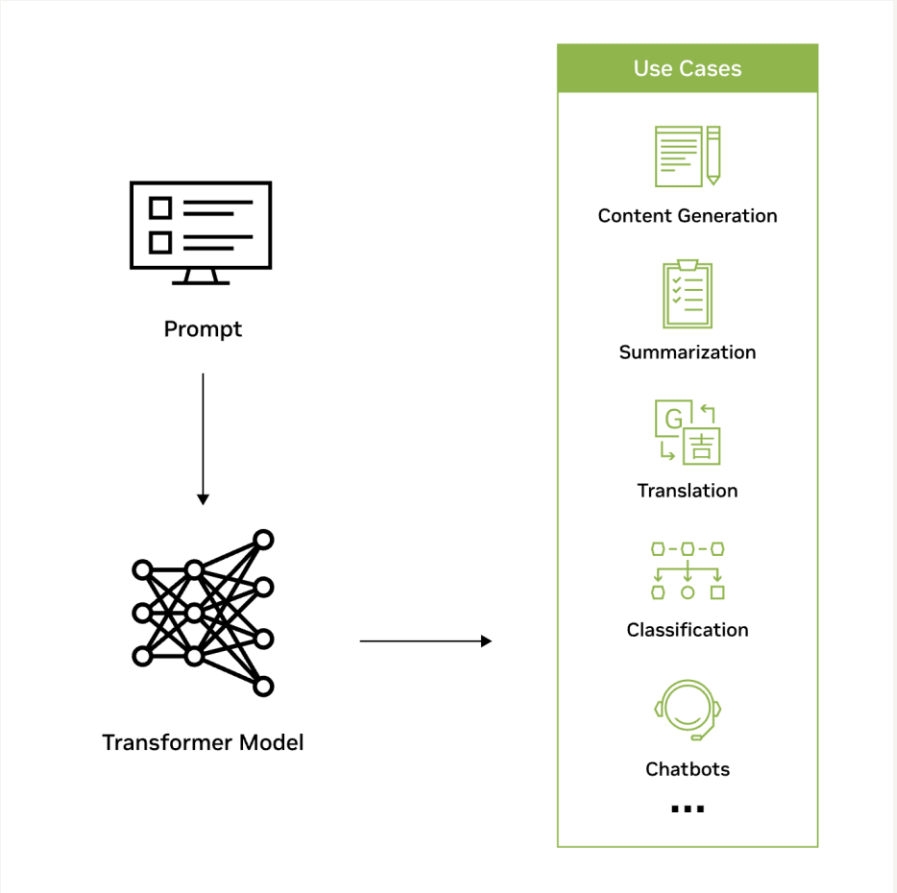
1. Extensive Data Preprocessing requirements
2. Dependency on Expert Knowledge
3. Complexity of Machine Learning Techniques
4. ML model deployment and maintenance

Large Language Models for Anomaly Detection

Large language models (LLMs) are deep learning algorithms (transformer networks) that can recognize, summarize, translate, predict, and generate content using very large datasets.

Benefits of using LLMs for AD:

- Streamlined data preprocessing
- Simplifying deployment of ML models
- Real-time monitoring
- Reduce the need for extensive expert knowledge



Large Language Models for Anomaly Detection

Use of pretrained LLMs to detect anomalies from log files generated during the execution of computational workflows.

pretrained LLM models – neural networks trained on massive text corpora to understand and generate human-like languages (eg. GPT, BERT)

https://huggingface.co/docs/transformers/index

Use of pretrained models in two way:

- Supervised fine tuning(SFT)
- In-context learning (ICL)

Large Language Models for Anomaly Detection

Supervised Fine-Tuning

- Adapts pretrained language models to specific tasks or domain
- Involves training on a smaller, labeled dataset specific to the target task
- Adjust the model's parameters to optimize performance on new task

Application in Anomaly Detection

- Logs from computational workflows are treated as sequence of sentences
- Fine-tuned models classify each sentence as "normal" or "anomalous"

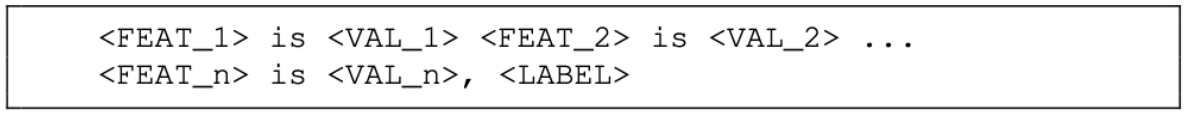


Fig. 2. Template of parsed log into a sentence.

Large Language Models for Anomaly Detection

In-context Learning

- LLMs perform tasks by leveraging examples provided in the query context
- Relies on prompt engineering instead of supervised fine-tuning
- Prompts includes task description, examples, and contextual cues to guide the model's output
- Useful in a scenario with scarce labeled data or for improving model generalization

Application in Anomaly Detection

- Prompt include job features, brief statistics, and labeled examples (e.g., normal or anomalous jobs)

Techniques

Zero-Shot prompts: Task description only, no examples

One-Shot prompts: Task description with a single example

Few-Shot prompts: Task description with a few examples

Large Language Models for Anomaly Detection

In-context Learning

```
# Prompt of task for ICL
You are a system administration bot.
Your task is to assess a job description with a couple
of features into one of the following categories:
Normal and Abnormal

You will only respond with the category.
Do not include the word "Category".
Do not provide explanations or notes.
A single job includes <FEAT_1> ... <FEAT_n>

# Example prompt
Instruct: <FEAT_1> is <VAL_1>, ... <FEAT_n> is <VAL_n>
Category: Normal
```

Fig. 3. Template of in-context learning.

Large Language Models for Anomaly Detection

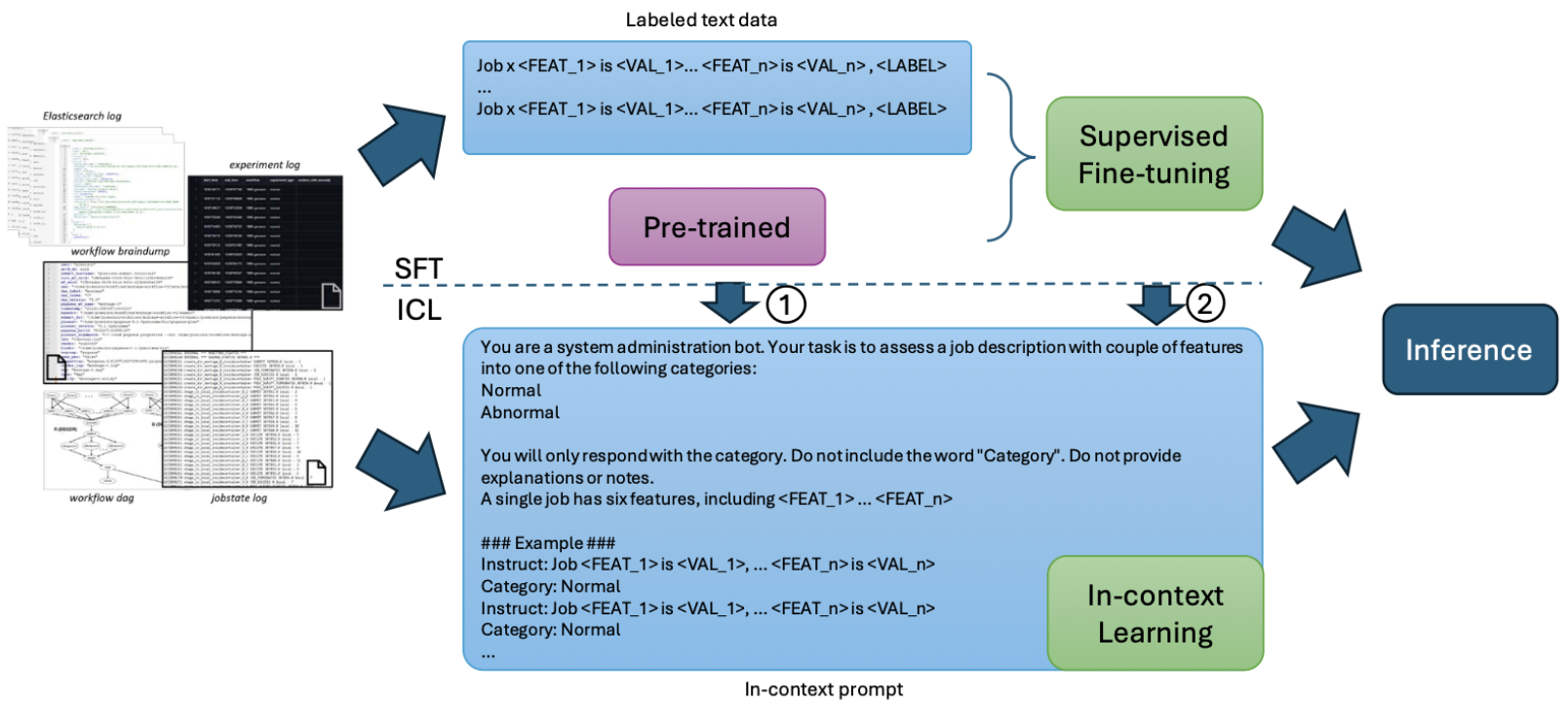


Fig. 1. Supervised fine-tuning and in-context learning for anomaly detection.

EXPERIMENT

Experiment

Dataset:

Flowbench, a collection of three computational workflows for anomaly detection

- 1000-Genome Workflow
- Montage Workflow
- Predict Future Sales Workflow

DATASET STATISTICS

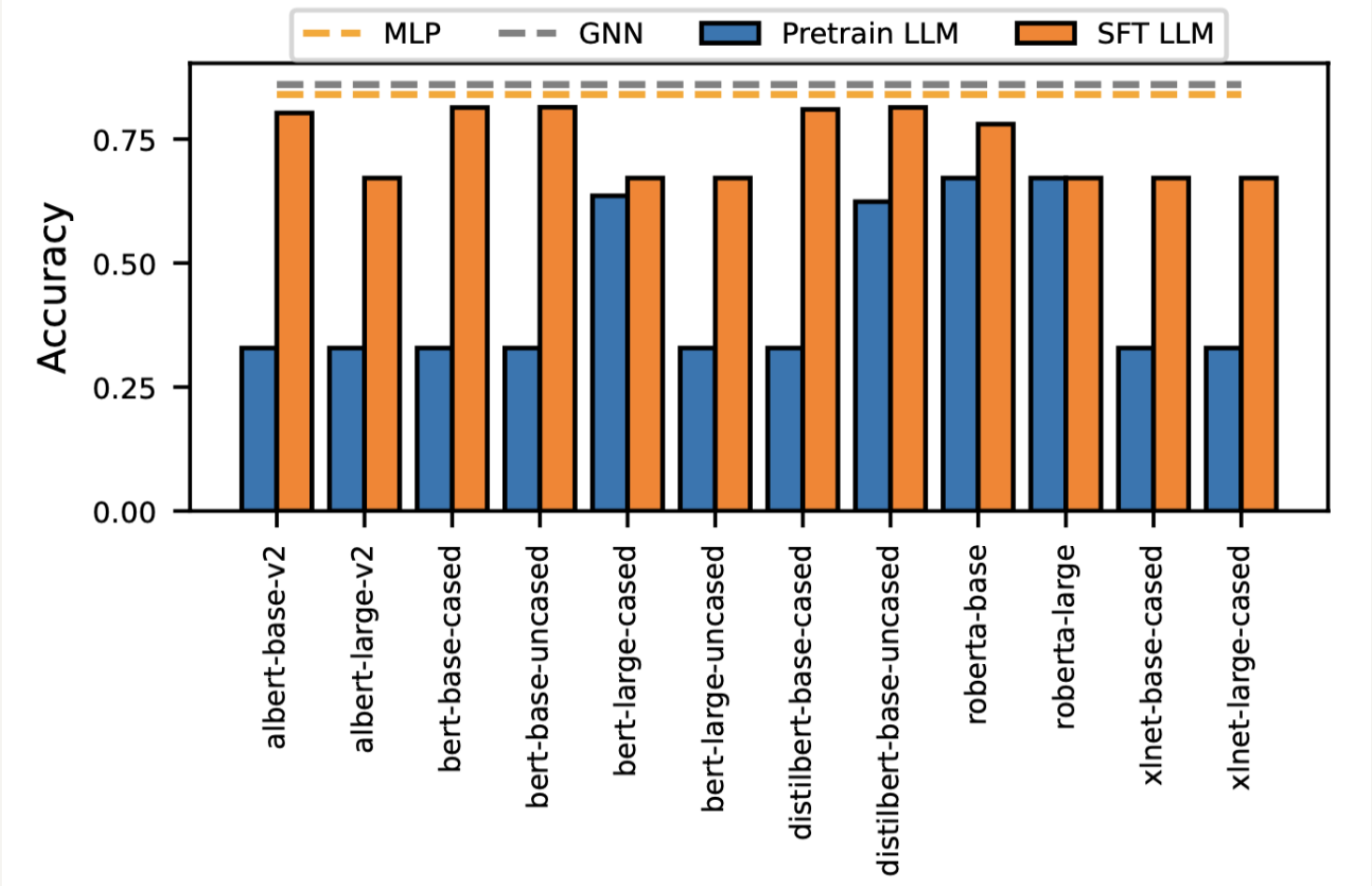
Dataset	Split	# of normal nodes	# of anomalous nodes	% of anomalies
1000 Genome	training	25911	12558	0.3264
	validation	3258	1551	0.3225
	test	3229	1580	0.3286
Montage	training	109738	28246	0.2047
	validation	13735	3513	0.2037
	test	13756	3492	0.2025
Sales Prediction	training	58043	13237	0.1857
	validation	7250	1660	0.1863
	test	7316	1594	0.1789

Data Preprocessing:

- Logs converted into tabular format
- Each row represent log entry
- Column consists of fields llike timestamps, job status, time duration, I/O operations

Experiment: SFT Models

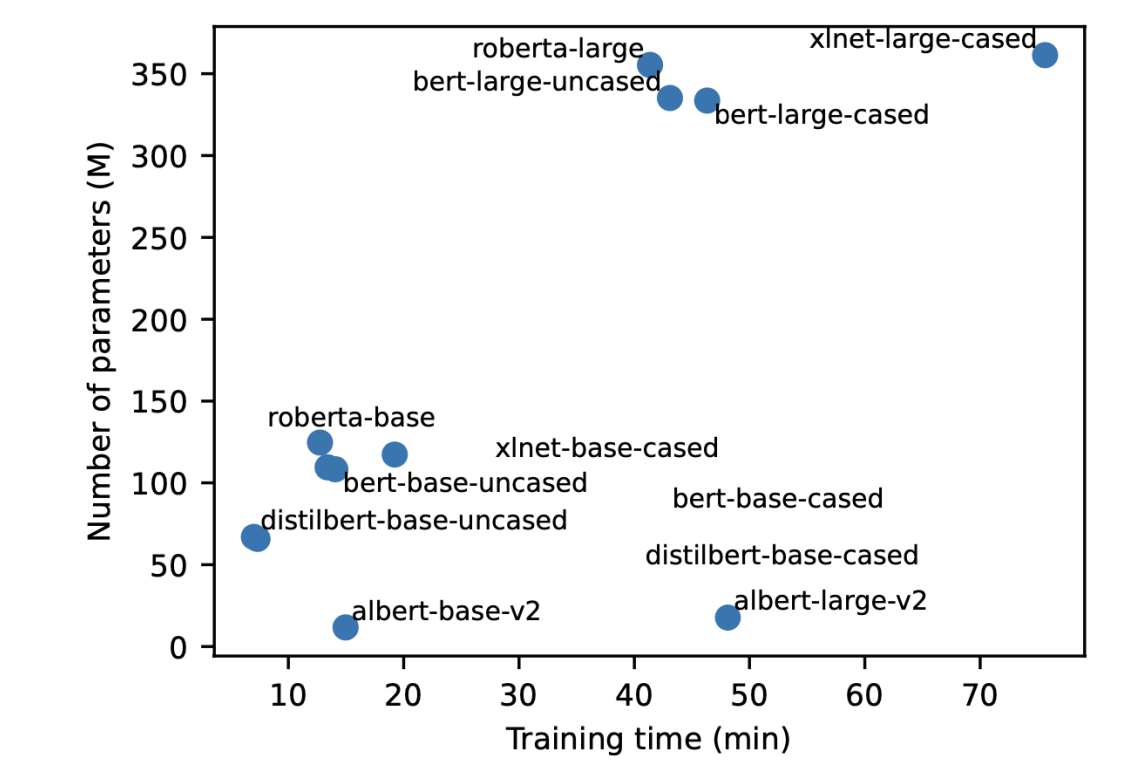
Accuracy comparison after supervised fine tuining on each pretrained LLM models.

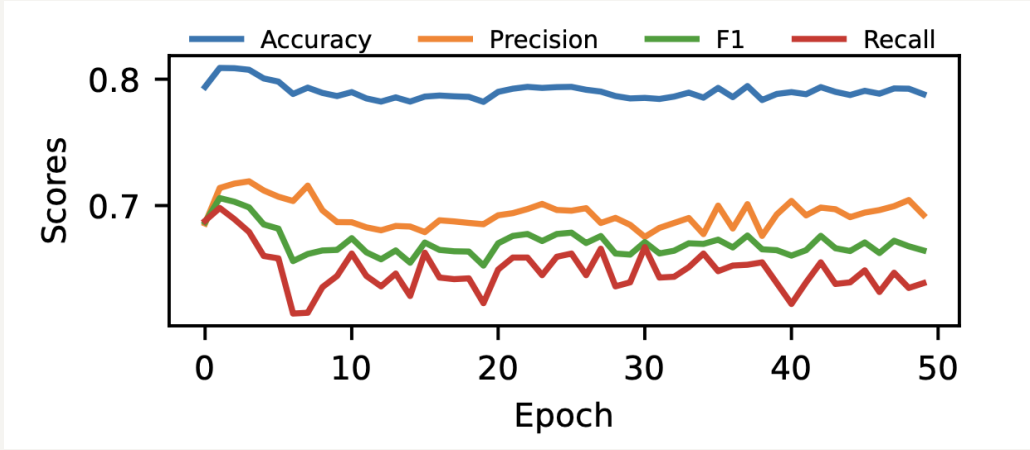


- SFT models outperform pretrained models and the performance is comparable to ML models.

Experiment: SFT Models

Relationship between model size and performance





- Training time per epoch = 260 seconds

- Performance of SFT models does not necessarily increase with number of parameters

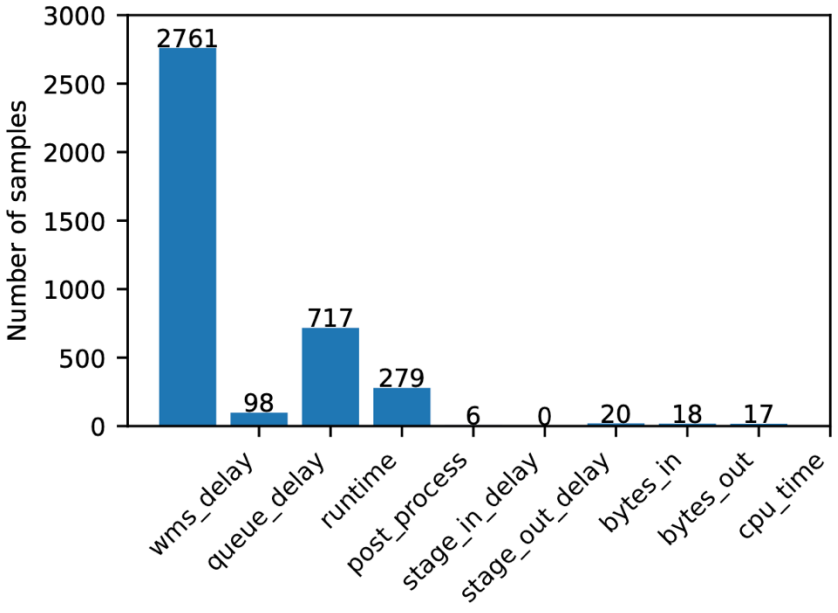
Experiment: Online Detection

Identifying system failures in real time

```
T1: wms_delay is 6.0
   ==> label: LABEL_0, score: 0.7708
T2: wms_delay is 6.0 queue_delay is 22.0
   ==> label: LABEL_0, score: 0.8103
T3: wms_delay is 6.0 queue_delay is 22.0 runtime is 2090.0
   ==> label: LABEL_0, score: 0.6631
T4: wms_delay is 6.0 queue_delay is 22.0 runtime is 2090.0 post_script_delay is 5.0
   ==> label: LABEL_1, score: 0.5780
T5: wms_delay is 6.0 queue_delay is 22.0 runtime is 2090.0 post_script_delay is 5.0 stage_in_delay is 1310.0
   ==> label: LABEL_1, score: 0.5742
```

Early Detection

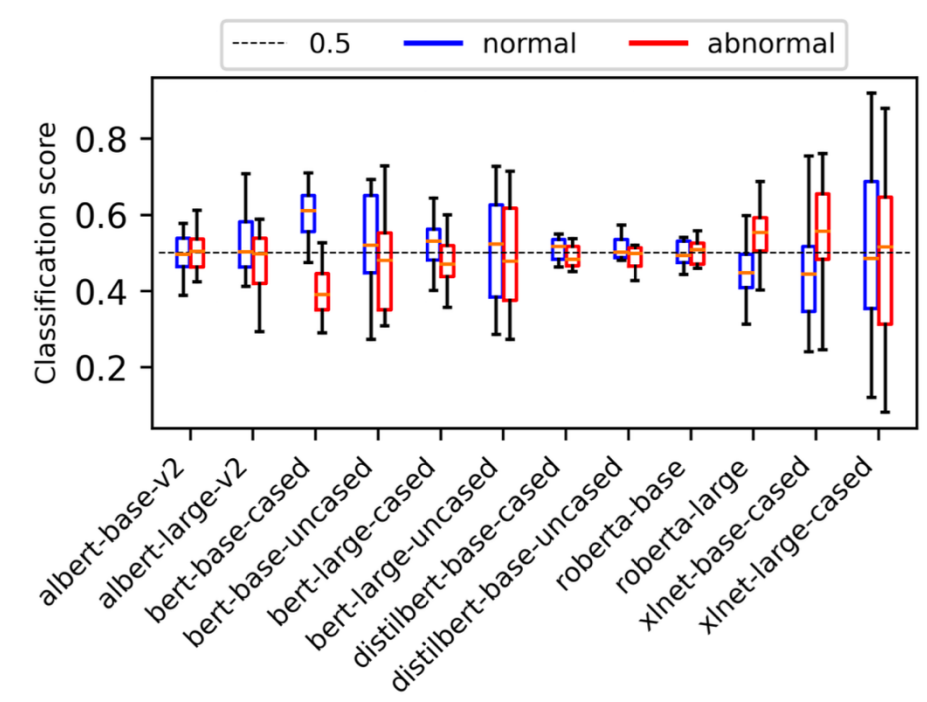
- Checking the first time model predicts a correct label for job
- SFT models can detect anomalies at the early stages of a job



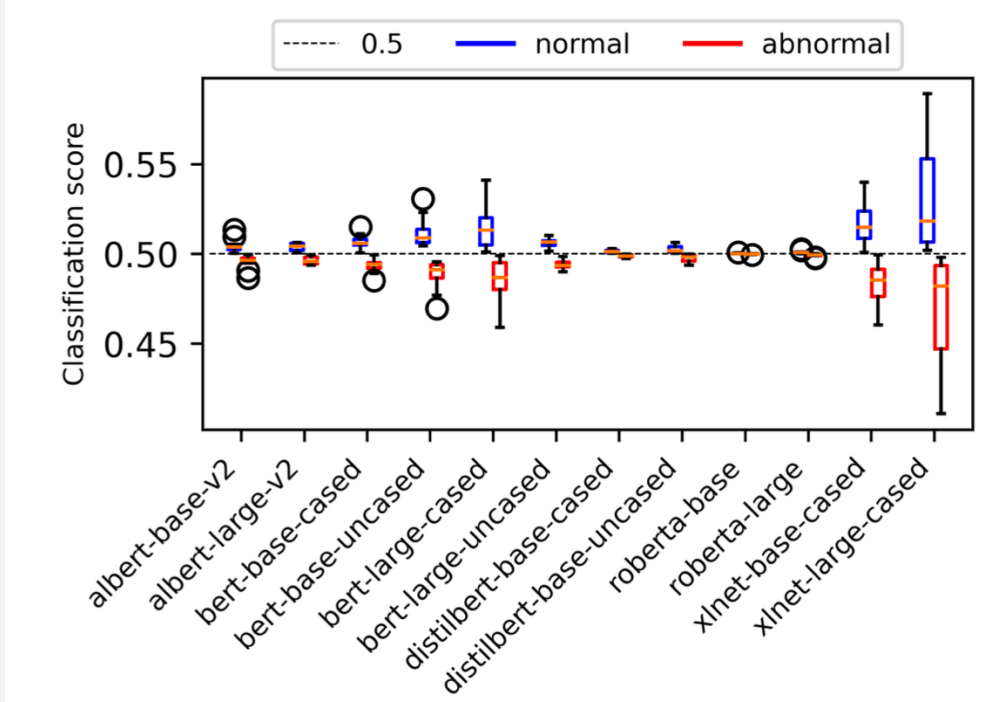
Experiment: Debiasing LLMs

Source of bias in SFT: Pretrained LLM itself, dataset utilized for fine-tuning

Ideally, for empty sentence, the model should predict normal and abnormal jobs with almost equal probability



Data Augmentation



Prediction is biased either toward normal or anomalous

Gap between normal and anomalous prediction is reduced

Experiment: Transfer Learning

Transfer learning: knowledge learned from one task to improve performance on another related task

Underlying hidden features learned from one dataset can be generalized to another dataset

Experiment: Overcoming Catastrophic Forgetting

Catastrophic Forgetting: occurs when model forgets previously learned information as it learns new information, a common problem in SFT models

- Solution: Freeze the parameters of the model that were learned during pretraining and update only the parameters that are specific to new task

FREEZING PARAMETERS. PRETRAINED MODEL: BERT-BASE-UNCASED.

	SFT (D1)	SFT (D1 + D2)	SFT (D1 + D2)
Param. updated	All	All	Linear
Accuracy	0.8155	0.7065	0.7305
Precision	0.7251	0.5589	0.9150
Training time (sec.)	801	2849	314

Training by freezing pre-trained parameters

Experiment: In-context learning

ICL is a text generation task, so decoder-only models are used

To reduce training resource and time, following two techniques are applied

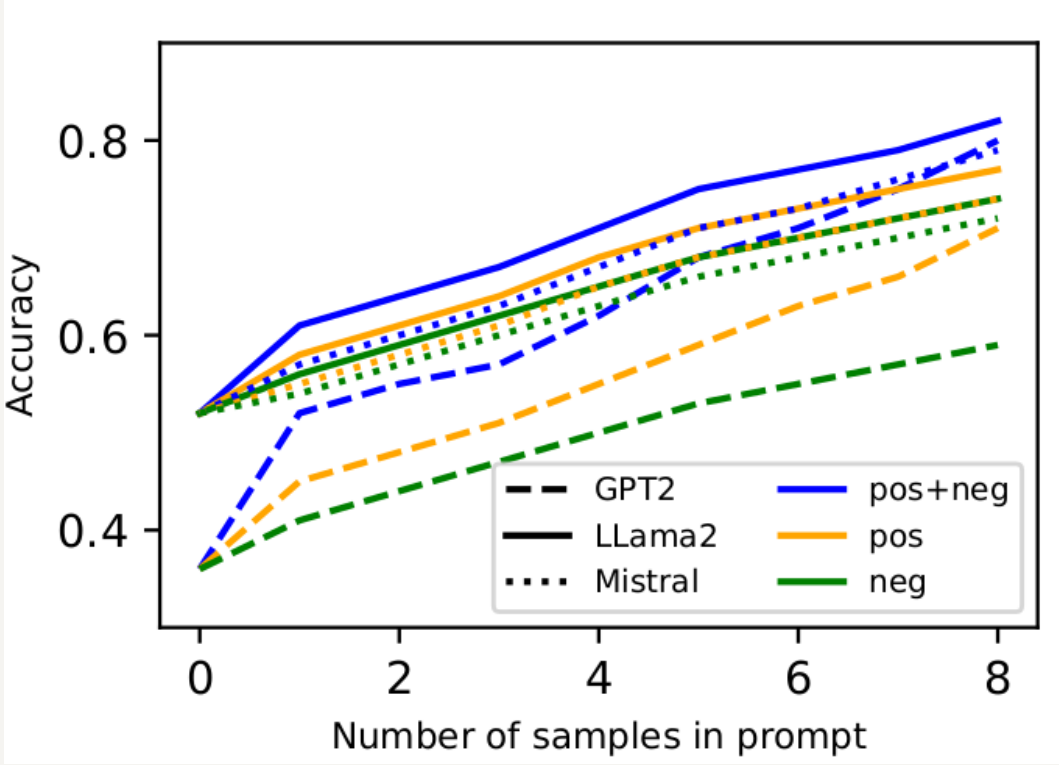
- Quantization: converting the model's weights from floating-point to fixed point numbers
- Low Rank Adaptation (LoRA): Instead of updating all the parameters during fine-tuning, LoRA modifies only a small subset of parameters.

Accuracy of ICL models on 1000 Genome dataset:

Model	All param.	LoRA param(%)	FT	Few-shot (neg. only)	Few-shot (pos. only)	few-shot
GPT2	127 M	2 M (1.86%)	No	0.54	0.57	0.66
			Yes	0.68	0.73	0.72
Mistral	7 B	27 M (0.38%)	No	0.64	0.65	0.68
			Yes	0.73	0.68	0.78
LLama2	7 B	34 M (0.50%)	No	0.60	0.63	0.65
			Yes	0.68	0.71	0.76

Experiment: In-context

Model's performance with increasing number of samples



GPT2 are more applicable under ICL because they require only a few examples to achieve a similar performance compared with large models such as Mistral-7B and LLama2-7B.

Experiment: Chain of Thought (CoT) Reasoning

CoT: breaking down the decision-making process into series of logical steps where the ICL models generate a sequence of intermediate steps.

ICL models become more interpretable and trustworthy

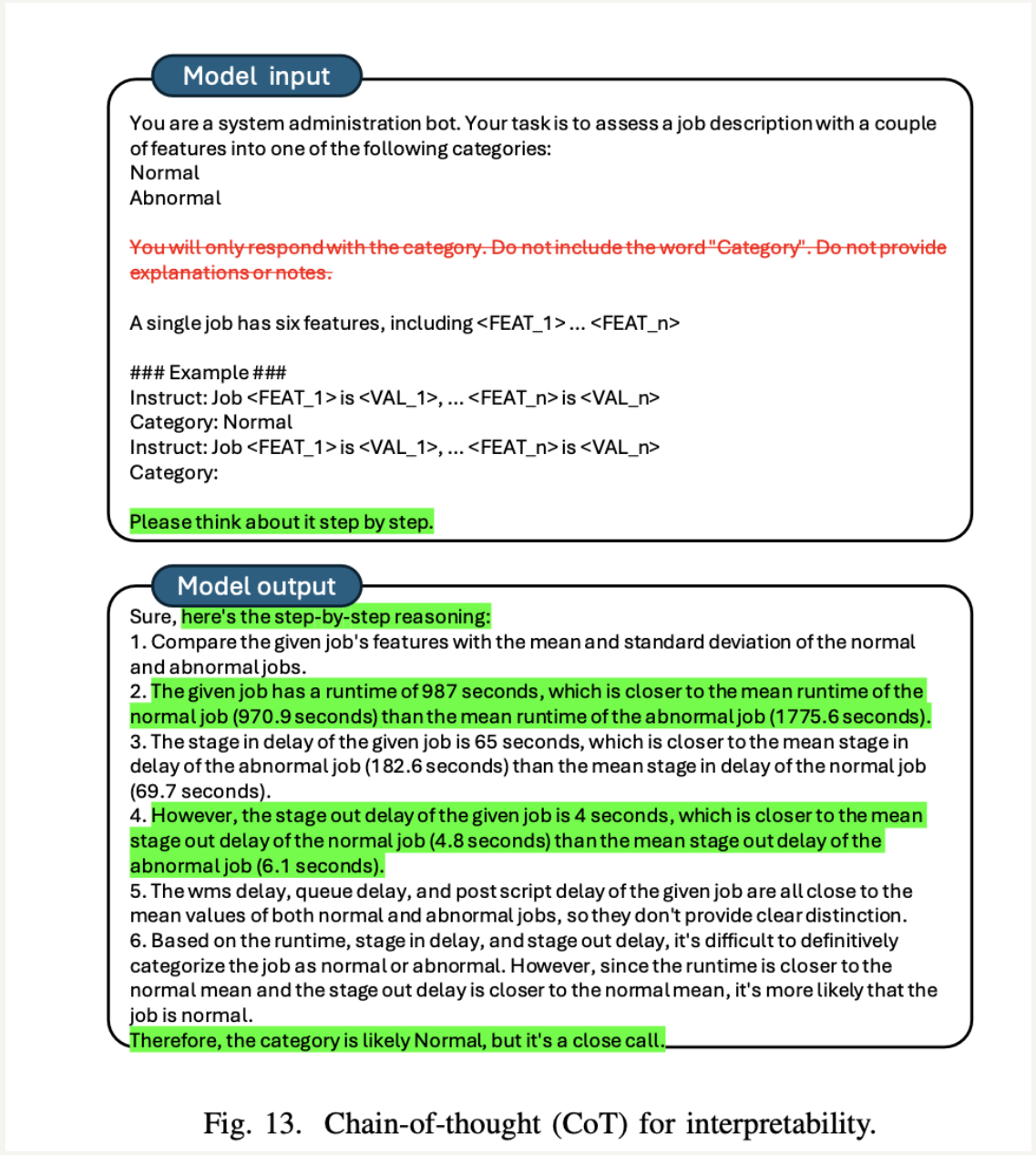


Fig. 13. Chain-of-thought (CoT) for interpretability.

Conclusion

- LLMs demonstrate effective anomaly detection using supervised fine-tuning (SFT) and in-context learning (ICL).
- SFT achieves high accuracy with minimal labeled data and strong generalization via transfer learning.
- ICL enables flexible few-shot and zero-shot detection using prompt engineering without fine-tuning.
- Advancements in LLM optimization will enhance their scalability and applicability in real-time anomaly detection.

Limitations

- Requirement of the labeled dataset which is a challenge in computational workflow
- Does not explore how proposed methods perform on large and complex workflow
- Feasibility of deploying the methods for real time anomaly detection in high throughput environments is not disscussed throughly.

Experiment Artifacts

Flow-Bench: A Dataset for Computational Workflow Anomaly Detection

Flow-Bench is a benchmark dataset for anomaly detection techniques in computational workflows. Flow-Bench contains workflow execution traces, executed on distributed infrastructure, that include systematically injected anomalies (labeled), and offers both the raw execution logs and a more compact parsed version. In this GitHub repository, apart from the logs and traces, you will find sample code to load and process the parsed data using pytorch, as well as, the code used to parse the raw logs and events.

Dataset

The dataset contains 6352 DAG executions from 11 computational science workflows and 1 ML data science workflow, under normal and anomalous conditions. These workflows were executed using Pegasus WMS - Panorama. Synthetic anomalies, were injected using Docker's runtime options to limit and shape the performance. The data have been labeled using 6 tags (normal, cpu_2, cpu_3, cpu_4, hdd_5 and hdd_10).

- *normal*: No anomaly is introduced - normal conditions.
- *CPU K*: M cores are advertised on the executor nodes, but on some nodes, K cores are not allowed to be used. (K = 2, 3, 4M = 4, 8 and K < M)
- *HDD K*: On some executor nodes, the average write speed to the disk is capped atK MB/s and the read speed at (2×K) MB/s. (K = 5, 10)