ITARODI: WHAT MAKES A GOOD DISTRIBUTED TRACE REPRESENTATION?

Anonymous authors
Paper under double-blind review

ABSTRACT

Distributed traces—graph-based records of individual request executions—are critical for understanding and operating production software systems across industries, supporting security, performance optimization, and debugging. However, they have received comparatively little attention in machine learning (ML), despite rapid progress in both AI-driven software engineering and graph representation learning. We address this gap with **Iditarod**, the first multitask benchmark for distributed trace representation learning on real-world data. Iditarod is based around a new corpus of 3.18 million traces drawn from a commercial observability platform's own telemetry, combined with a subset of the Uber trace corpus. Its four tasks evaluate two complementary aspects of representation quality: (a) preserving key properties of individual traces through two intrinsic tasks and (b) providing efficient and performant embeddings for downstream applications through the Trace Retrieval task and the multimodal Faulty Deployment detection task over traces and time series. Our evaluation of sixteen different trace embedding methods on Iditarod includes models that process traces as either serialized text, call-path collections, or graphs. We find that to match or outperform language encoders on the intrinsic tasks and Trace Retrieval, graph neural networks often need to exploit the directed acyclic structure of traces. On Faulty Deployment, meanwhile, we find that no encoder produces trace embeddings that improve upon using time series alone, while a handcrafted feature extractor does, revealing a gap in current graph representation capabilities. Iditarod's data and evaluation code are available under the Apache 2.0 License.

1 INTRODUCTION

Modern software systems are built on distributed architectures, decomposing applications into hundreds to thousands of independently deployable services to gain agility and scalability (Dragoni et al., 2017; Li et al., 2022a; Soldani et al., 2018). A single request may traverse dozens of these services across multiple nodes and datacenters (Li et al., 2022a; Zhou et al., 2021). System failures have been estimated to cost trillions of dollars each year, making rapid detection and diagnosis essential (Krasner, 2018; Zhang et al., 2023). To operate these systems, engineers rely on *observability platforms*, which center on three major signals: metrics, logs, and distributed traces (Li et al., 2022a). Each trace records a request's execution as a rooted tree of spans, each carrying timing and execution metadata and its parent-child position (Sigelman et al., 2010; Li et al., 2022a).

Traces are a natural fit for graph learning, and a growing body of work applies machine learning to them (Zhang et al., 2023; Ding et al., 2024; Liu et al., 2020; Xie et al., 2023a;b; Zhang et al., 2022a), resulting in benchmarks for anomaly detection and root cause analysis (Sun et al., 2025; Pham et al., 2025; Xu et al., 2025). However, these rely on faults injected into small-scale testbeds rather than incidents observed in production, which limits how well they reflect real-world complexity. Furthermore, they focus mainly on failure detection, a high-value use of trace data but not the only one. In particular, traces are also used to understand system behavior, e.g. by retrieving requests that match a pattern or inspecting an execution's structure; past benchmarks' focus on failure detection tells us little about such capabilities. We thus want *multitask* benchmarks, which in other modalities have been catalysts for progress in representation learning (Enevoldsen et al., 2025; Hu et al., 2020).

Overall, these limitations raise a basic question: what makes a good distributed trace representation? Our answer is that a good representation both (1) preserves important properties of individual traces

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

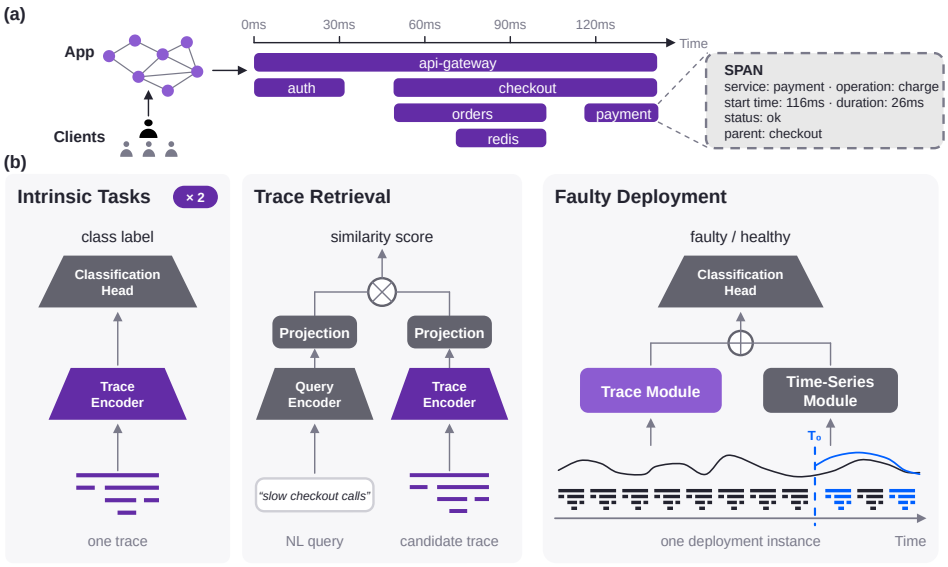


Figure 1: **One trace encoder, four tasks. (a)** A request traverses a distributed application, whose services emit spans that together form a trace, shown as a flame graph in which nesting encodes parent-child relations and bar length represents duration. **(b)** A trace encoder (purple) maps each trace to a fixed-size vector consumed by the two intrinsic tasks and by two extrinsic tasks, Trace Retrieval and Faulty Deployment. Task architectures operate on one trace, a query and candidate trace, or a deployment instance combining traces with APM time series.

and (2) supports useful downstream operational tasks by efficiently encoding the relevant information. To measure these complementary desiderata, we introduce **Iditarod**, a benchmark built on a new release of 3.18 million real-world traces drawn from a commercial observability platform's own telemetry combined with a subset of the Uber trace corpus (Lee et al., 2024). The following summarizes the contributions we make via the release of and evaluations with Iditarod:

- **The first real-world multitask trace benchmark:** Iditarod is the first machine learning benchmark for evaluating representation on real-world traces. It contains two intrinsic tasks, which measure where latency is concentrated and how errors are distributed, and two extrinsic tasks, which evaluate utility on practical workflows: trace retrieval from natural-language queries and faulty deployment detection. The latter task is both *multi-modal*, as we provide contemporaneous application performance monitoring (APM) time series, and *latency-sensitive*, requiring representation and prediction to happen fast enough to enable early remediation. In addition to being multi-modal, Iditarod is also *multi-source*, incorporating traces from the observability company's internal systems and the Uber corpus, and includes the public release of 3.18 million new traces.
- **A method for trace-retrieval labeling:** To build the trace-retrieval task, we introduced budgeted query relaxation, a method for deriving graded query–trace relevance labels at scale without human annotation. The approach is validated via human graders, achieving 94.8% agreement.
- **Diverse evaluation:** We evaluate sixteen trace encoders spanning distinct ways of processing traces: as text, call-paths, and graphs; for the latter, we also study different architectures that take varying advantage of edge structure, including set-based approaches, general graph approaches, and directed acyclic graph (DAG) approaches. Increasing exploitation of edge structure leads to graph neural networks (GNNs) that are competitive with large-language model (LLM) embedders, but neither is uniformly better across categories. Current encoders thus capture different aspects of trace information but do not provide a uniformly strong representation across both desiderata.
- **New capabilities gaps:** We find that all trace representation methods fail to improve upon using APM time series alone at faulty deployment detection. At the same time, by combining handcrafted features with quantile-based pooling, we show that traces contain useful information for this task. We also find that LLMs—whether used as embedders, in-context learners, or agents—are often too costly to use in practice for tasks like faulty deployment detection. Such gaps in capabilities establish concrete objectives for future trace representation methods to conquer.

Table 1: Comparison with graph benchmarks.

Benchmark	Domain(s)	#Graphs	#Nodes	#Edges	Temporal
OGB (2020)	Society, Nature, Information	1M+	229M+	2B+	✗
LRGB (2023)	Chemistry, Computer Vision	694K+	82M+	404M+	✗
TGB (2023)	Social, Interaction, Rating, Traffic, Trading	9	2M+	240M+	✓
Iditarod	Observability	3.2M+	240M+	237M+	✓

2 Related Work

Graph learning benchmarks. The graph ML community has built standardized benchmarks around canonical tasks such as node classification, link prediction, and graph classification (Hu et al., 2020; Dwivedi et al., 2023). Graph retrieval from natural-language queries remains underexplored in these benchmarks (see Appendix C.1). Recent position work argues that further progress requires benchmarks grounded in real-world applications and developed with domain expertise (Bechler-Speicher et al., 2025). Observability offers a rich set of real-world use cases yet remains underrepresented in graph-learning benchmarks (Table 1). In particular, distributed traces form rooted execution trees, a special case of DAGs. Despite the many architectures proposed for DAGs (Zhang et al., 2019; Thost & Chen, 2021; Dong et al., 2022; Luo et al., 2023; Rey et al., 2026), real-world benchmarks remain scarce, with evaluations relying heavily on synthetic DAGs or graphs made acyclic by removing edges (Luo et al., 2023). Beyond static graphs, temporal and dynamic graph learning has attracted growing interest, motivating benchmarks such as TGB (Huang et al., 2023). Traces offer a distinct setting for this because their temporal information is tied to events within an individual request execution rather than to graph evolution.

AIOps and trace analysis. Much of trace-based ML research has focused on anomaly detection and root cause analysis (Liu et al., 2020; Xie et al., 2023b; Ding et al., 2024; Pham et al., 2025). Open-source microservice testbeds such as TrainTicket (Zhou et al., 2021) and DeathStarBench (Gan et al., 2019) have supported this work by enabling controlled fault injection and data collection. However, results are difficult to compare across studies: some relied on non-public production data (Xie et al., 2023a; Zhang et al., 2023), while others used open-source testbeds with heterogeneous pipelines, fault configurations, and label definitions. Public releases of real-world distributed traces remain rare; the Uber corpus is one example, but it does not define benchmark tasks (Lee et al., 2024). Recognizing this fragmentation, recent efforts have established standardized benchmarks for trace analysis (Appendix D.1, Table 15). TADBench (Sun et al., 2025) consolidates several datasets under a unified framework for trace anomaly detection. RCAEval (Pham et al., 2025) provides 735 failure cases across three testbed systems with multi-source telemetry for root cause analysis. OpenRCA (Xu et al., 2025) curates 335 cases from AIOps Challenge datasets (Li et al., 2022b), including systems deployed on real-world infrastructure. Two limitations remain. First, the underlying systems span only tens of services, limiting the scale and topological diversity represented. Second, their anomalies are predominantly resource-level faults such as CPU stress, memory exhaustion, network delay, and packet loss, introduced through chaos engineering tools rather than encountered naturally in production. Faulty deployment detection lies outside this scope. It asks whether a new service version introduced a fault. To our knowledge, no public benchmark targets this task or pairs real-world traces with labels from naturally occurring deployment faults.

Together, these gaps motivate a benchmark that evaluates trace representations across both intrinsic properties of individual executions and downstream operational tasks, using real-world traces rather than exclusively controlled fault-injection data.

3 Iditarod: A Trace-Centric Multitask Benchmark

Iditarod combines two sources of real-world traces. The first, which we release under the procedure described in Appendix A, consists of 3.18 million traces sourced solely from a leading commercial observability platform's own monitoring of internal systems, free of any customer data. We also integrate a subset of the Uber trace corpus (Lee et al., 2024), yielding data from two independent systems. Iditarod defines four tasks in two families, intrinsic and extrinsic. Both corpora contribute traces to the intrinsic tasks, while the extrinsic tasks draw only on the observability-company corpus. Intrinsic tasks test whether a representation preserves key properties of an individual trace, indepen-

Table 2: **Query taxonomy.**

Category	Description	Example
Trace-Level	Aggregate trace properties only.	*Show me traces that took more than a second.*
Non-Structural	One or more span constraints, no relation required between them.	*Requests that touched both service A and service B.*
Structural	At least one relation between spans, direct or indirect.	*Which traces have service A calling into service B?*

dently of any downstream workflow. The two together comprise 54,755 labeled examples and probe complementary combinations of span attributes and graph structure:

- **Latency Bottleneck (LB)** combines duration and critical-path structure. Its labels are *has bottleneck* and *no bottleneck*. A trace has a bottleneck when one non-root critical-path span contributes strictly more exclusive time than all remaining critical-path spans combined.
- **Error Pattern (EP)** combines span status and structure. It distinguishes traces with *no error*, *isolated* error spans, and *propagated* errors along a parent-child chain.

Label constructions, dataset statistics, and an illustration appear in Appendix E. Two extrinsic tasks measure utility on real workflows: Trace Retrieval (TR) and Faulty Deployment Detection (FD).

3.1 IDITAROD-TR: TRACE RETRIEVAL

Runtime conditions can make nominally similar requests follow very different execution paths, so even the engineers who own a service struggle to anticipate the full space of traces it produces. Retrieving traces is therefore central to understanding how a system behaves in production and to inspecting requests during incidents. Observability platforms expose structured query languages that retrieve traces matching explicit predicates and are effective for exact filtering. Embedding-based retrieval complements them by ranking semantically related traces when the behavior of interest is only partially specified or expressed in natural language. It supports trace search without mastering a platform-specific query language and surfaces relevant traces beyond exact matches. Compact trace representations also offer an attractive hot storage layer in this setting: on our data, a 512-dimensional float32 embedding is 23$\times$ smaller than the average trace serialized as JSON. These applications motivate Iditarod-TR. Given a natural-language query q_{nlq}, TR evaluates rankings of its labeled candidate traces by relevance (Figure 2). Table 2 summarizes the three query types.

Producing graded relevance judgments at this scale by hand would require experts to assess many query–trace pairs (Thakur et al., 2021). We instead introduce **budgeted query relaxation**: starting from exact matches, the method progressively removes or weakens query constraints, and relevance decreases with the number of changes needed to retrieve a trace (Figure 2). Each q_{nlq} is an LLM rendering (see Appendix C.4) of a structured trace query q_{struct} executed by a deterministic query engine. For concision, we write q for q_{struct} and let $F(q)$ denote the set of traces returned by q. Controlled relaxations of q expand this set, yielding graded relevance labels without human judgments on individual pairs. Budgeted query relaxation has two components: an atomic relaxation (one step of query generalization) and a budget multiplier $B \in \mathbb{N}^*$, the number of atomic relaxations between adjacent relevance levels. With four levels, a path spans a budget of $3B$ atomic relaxations. We formalize both below, starting from the query structure that relaxations operate on.

Query structure. A structured query decomposes as $q = TQ \wedge TF$. The *trace-level filter* TF conjoins constraints on trace attributes such as `total_duration` and `total_span_count`. The *trace query* TQ combines aliases and relations. Each alias $a = s \wedge \left(\bigwedge_{i=1}^{k} t_i\right)$ pairs a scope $s \in \{\text{ROOT}, \text{ENTRY}, \text{ANY}\}$ with tag constraints $t_i = (\text{KEY}_i, \text{OP}_i, \text{VALUE}_i)$ comparing an attribute to a literal. The three scopes select, respectively, the root, a service entry span, or any span. TQ is a disjunction of conjunctions of alias and relational literals encoding ancestor-descendant relations.

Relaxations. An *atomic relaxation* is a local transformation a satisfying $F(q) \subset F(a(q))$. Examples include dropping a tag, relaxing a numerical threshold, widening a scope, and weakening a relational operator. The complete catalog and its monotonicity-preserving preconditions appear in Appendix C.3. Applying T atomic relaxations yields a path $P_j = (q_j^{(0)}, \ldots, q_j^{(T)})$, where $q_j^{(0)} = q$

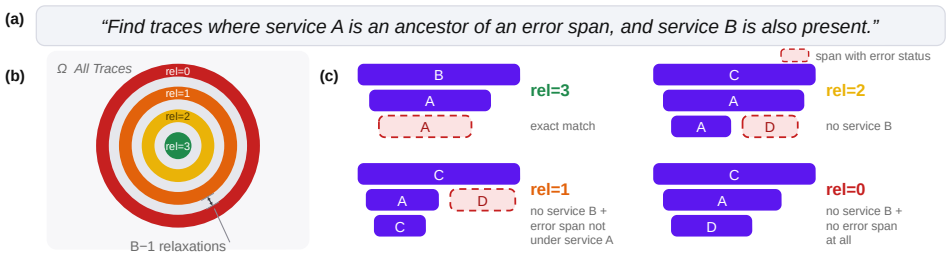


Figure 2: **Budgeted query relaxation.** **(a)** A natural-language query. **(b)** Nested result sets from relaxation. Rings are relevance levels, spaced by budget B. **(c)** One trace example per level.

and $F(q_j^{(t)}) \subset F(q_j^{(t+1)})$. Because several actions may be admissible at each step, we sample N independent paths. Each path defines four disjoint shells $S_k^{(j)} = F(q_j^{(kB)}) \setminus F(q_j^{(kB-1)})$ for $k \in \{0,1,2,3\}$. We set $F(q_j^{(-1)}) = \emptyset$, making $S_0^{(j)} = F(q)$ the exact-match shell, and assign $\mathrm{rel}_j(q,x) = 3-k$ to every $x \in S_k^{(j)}$. Thus, B controls the spacing between relevance levels. For a trace occurring along multiple paths, its final relevance is $\mathrm{rel}(q,x) = \max_{j \in [\![1,N]\!]} \mathrm{rel}_j(q,x)$.

The structured queries are drawn from the observability company's routine use of its own platform. Iditarod-TR contains 3,980 queries and 1,134,621 graded query–trace pairs. Observability engineers, blind to the relevance labels and shown only a natural-language query and two traces, judged 360 trace pairs from 90 queries. Their preferences match the label ordering on 94.8% of comparisons, including 94.2% of adjacent-level comparisons ($\Delta\,\mathrm{rel} = 1$), and pairwise inter-annotator agreement is 97.7%. The full protocol and per-dimension results appear in Appendix C.5.

3.2 Iditarod-FD: Faulty Deployment Detection

Software deployments are a major source of production incidents (Ghosh et al., 2022; Li et al., 2020). Once a new service version begins receiving traffic, its effects may surface in traces or APM time series as new call paths, altered trace structures, or shifts in error rate. Iditarod-FD therefore frames early faulty-deployment detection as a multimodal task over graph-structured traces and time series. The post-deployment window provides a natural axis of difficulty: shorter windows expose less evidence and require efficient inference, but they enable earlier intervention. Each Iditarod-FD instance is the deployment of a target version for a *service-resource*, a $(\text{service}, \text{operation}, \text{resource})$ triplet. Versions are deployed at the service level, but their effects can be localized to an operation or resource. Let T_0 denote the deployment time. Traces and aligned APM time series are collected over a pre-deployment window $[T_0 - W_{\text{pre}}, T_0)$, with $W_{\text{pre}} = 1\text{h}$, and a post-deployment window $[T_0, T_0 + W_{\text{post}}]$. Observations are partitioned by time and version into three cells: *baseline-pre* holds previous-version observations before T_0, while *baseline-post* and *new-post* hold concurrent post-T_0 observations from, respectively, previous versions and the target version. Figure 11 (Appendix D) illustrates this construction.

Constructing ground truth requires distinguishing deployment faults from transient or unrelated changes. Rollback alone is an insufficient proxy, since not every rollback indicates a fault and some faults are fixed by a later deployment instead. Fault labels come from the platform's deployment-monitoring pipeline. A deployment is labeled faulty only when three properties are jointly observed: significant impact relative to the service-resource baseline, temporal correlation with the target-version deployment, and persistence rather than transience. Because such evidence must accumulate, the labels are finalized one to three hours after deployment. Iditarod-FD evaluates prediction of these labels from only the first seconds or minutes of post-deployment observations.

We define four early-detection subtasks with $W_{\text{post}} \in \{30\text{s}, 1\text{m}, 5\text{m}, 10\text{m}\}$. All four share the same instances and train/test split, differing only in the amount of post-deployment evidence. Iditarod-FD contains 1,158 labeled instances, 14.6% of which are faulty.

4 Task Architectures and Trace Encoders

We first present the task-specific architectures that consume fixed-length trace representations, then the encoder families that map different input views to such representations.

4.1 TASK-SPECIFIC ARCHITECTURES

Let x denote a trace and $f_\theta(x) \in \mathbb{R}^d$ its representation. Figure 1 outlines how $f_\theta(x)$ is used across tasks: intrinsic tasks classify one trace, Trace Retrieval compares a query with a candidate trace, and Faulty Deployment combines bags of traces with APM time series. For each intrinsic task k, a head g_k maps $f_\theta(x)$ to class logits and is trained with cross-entropy.

Trace retrieval. We use a two-tower dual encoder, a standard dense-retrieval architecture that lets candidate traces be indexed independently of the query (Thakur et al., 2021). A language encoder f_q maps the query q_{nlq}, while f_θ maps a candidate trace x. When the trace is serialized as text, both towers share the same language encoder and projection. Otherwise they are asymmetric. Projection heads P_q and P_x map both outputs to a common space. Cosine similarity gives the retrieval score

$$s(q_{\text{nlq}}, x) = \frac{P_q(f_q(q_{\text{nlq}}))^\top P_x(f_\theta(x))}{\|P_q(f_q(q_{\text{nlq}}))\|_2 \, \|P_x(f_\theta(x))\|_2}. \tag{1}$$

We train the dual encoder with a pairwise margin-ranking loss on candidate pairs (x^+, x^-) whose graded relevance labels satisfy $\text{rel}(q_{\text{struct}}, x^+) > \text{rel}(q_{\text{struct}}, x^-)$. The per-pair objective is $\max(0, m - s(q_{\text{nlq}}, x^+) + s(q_{\text{nlq}}, x^-))$. Further details appear in Appendix C.

Faulty deployment. We introduce a three-cell multimodal framework over the baseline-pre, baseline-post, and new-post cells of Section 3.2. Its historical and concurrent references help distinguish version-specific changes from temporal shifts affecting both versions. As detailed in Figure 12 (Appendix D), a trace branch operates on bags of traces and a time-series branch on the APM time series. Their outputs h_T and h_{TS} are concatenated and passed to a binary classification head. For the trace branch, let $\mathcal{C} = \{\text{baseline-pre}, \text{baseline-post}, \text{new-post}\}$ and $\mathcal{X}_c = \{x_{c,n}\}_{n=1}^{N_c}$ denote the bag of N_c traces in cell $c \in \mathcal{C}$. A shared encoder and per-trace projection independently map every trace to $z_{c,n} = P_x(f_\theta(x_{c,n})) \in \mathbb{R}^p$. The trace branch has two replaceable aggregation stages: a *set pool* summarizes each cell's unordered bag, and a *cross-cell aggregator* combines the summaries. Because a fault may affect only a subset of requests, in which case its signal can be concentrated in the tails of the trace-embedding distribution, our default choice for these two stages, *tail-exceedance*, uses order statistics to compare each post-deployment cell with a two-sided baseline-pre reference. For coordinate j, let ℓ_j and u_j denote the empirical quantiles of the baseline-pre embeddings at levels α_ℓ and α_u, respectively. For each post-deployment cell $c \in \{\text{new-post}, \text{baseline-post}\}$, tail-exceedance computes the average of the lower- and upper-tail exceedance rates,

$$r_{c,j} = \frac{1}{2N_c} \sum_{n=1}^{N_c} \left(\mathbb{1}[z_{c,n,j} < \ell_j] + \mathbb{1}[z_{c,n,j} > u_j] \right). \tag{2}$$

Both tails receive equal weight because the orientation of a learned embedding coordinate is arbitrary. Writing $r_c = (r_{c,1}, \ldots, r_{c,p})$, the trace block is $h_T = [r_{\text{new-post}} \,\|\, r_{\text{baseline-post}}]$. Further details on the classifier architecture and training procedure appear in Appendix D.

4.2 TRACE ENCODER FAMILIES

Our evaluation covers three views of a trace: serialized text, call-path collections, and graphs. Preprocessing, parameter counts, and configuration details appear in Appendix F. In addition to the encoders specified below, on the faulty deployment task we also evaluate a novel *handcrafted* representation, which is a parameter-free 25-dimensional trace vector with features summarizing span-depth and parent-span-type distributions (cf. Appendix D for details).

Text. We serialize traces as JSON lists for MiniLM (Wang et al., 2020; Reimers & Gurevych, 2019), EmbeddingGemma-300M (Gemma 3) (Schechter Vera et al., 2025), and Qwen3-Embedding-4B (Zhang et al., 2025). The pretrained encoders remain frozen during task-specific training, testing whether text embeddings recover useful trace properties without explicit graph inductive bias.

Call-path collections. Two established trace-analysis baselines summarize call paths. The Service Trace Vector (STV) indexes observed call paths and associates each coordinate with latency information (Liu et al., 2020). The Service Critical Path Vector (SCPV) retains call paths traversed by critical-path spans, weighted by exclusive execution time (Zhang et al., 2022b).

Graph. We study three levels of access to the trace graph, ranging from no edges to general graph connectivity and DAG structure. This comparison tests the value of increasingly specific structural

Table 3: Overall benchmark leaderboard. Ranks among the 16 trace encoders, lower is better. **bold** = best, underline = second-best.

		Task rank				Aggregate rank		
Input formats	Encoder	LB AUPRC	EP Macro AUPRC	TR MAP@10	FD AUPRC	Intrinsic Avg. rank	Extrinsic Avg. rank	Overall Avg. rank
Text	Qwen3-4B	6.00	6.25	4.00	8.00	6.12	6.00	6.06
	Gemma 3	13.00	10.50	6.00	6.25	11.75	6.12	8.94
	MiniLM	8.00	12.50	13.00	6.25	10.25	9.62	9.94
Call-path	SCPV	11.50	15.38	10.67	8.75	13.44	9.71	11.57
	STV	11.50	15.62	10.33	8.75	13.56	9.54	11.55
Set	Deep Sets	8.00	9.00	6.67	5.00	8.50	5.83	7.17
	Set Transformer	9.50	9.00	5.67	2.75	9.25	**4.21**	6.73
Graph	GIN	7.00	9.50	5.67	5.50	8.25	5.58	6.92
	GCN	10.75	7.50	5.67	13.00	9.12	9.33	9.23
	GATv2	11.25	4.25	10.67	11.75	7.75	11.21	9.48
	Graphormer	**3.25**	5.25	7.33	14.75	4.25	11.04	7.65
	SAN	9.50	11.75	13.67	14.50	10.62	14.08	12.35
	GT	13.75	11.25	13.33	**1.50**	12.50	7.42	9.96
DAG	DAGNN	3.75	**1.00**	**1.67**	10.25	**2.38**	5.96	**4.17**
	PACE	6.00	2.25	6.00	13.00	4.12	9.50	6.81
	DAGformer	**3.25**	5.00	15.67	6.00	4.12	10.83	7.48

priors across tasks (Bechler-Speicher et al., 2025). Deep Sets and Set Transformer ignore parent–child edges and process each trace as an unordered span set (Zaheer et al., 2017; Lee et al., 2019). Deep Sets pools independently transformed spans, while Set Transformer models interactions within the set through attention. General graph encoders receive the trace as a graph with explicit span connectivity. They include the message-passing models GCN, GATv2, and GIN (Gilmer et al., 2017; Kipf & Welling, 2017; Brody et al., 2022; Xu et al., 2019), and the Graph Transformers GT, Graphormer, and SAN (Dwivedi & Bresson, 2021; Ying et al., 2021; Kreuzer et al., 2021). Finally, because a trace is a rooted tree and therefore a DAG (Sigelman et al., 2010; Li et al., 2022a), DAGNN, PACE, and DAGformer provide the strongest structural prior by exploiting partial order, reachability, or depth (Thost & Chen, 2021; Dong et al., 2022; Luo et al., 2023).

5 Experiments

We compare the 16 trace encoders across Iditarod's four tasks using task-specific training runs and a common trace-vector interface (Figure 1). Within each task, we vary the encoder while retaining the surrounding task architecture. All experiments use five seeds. Complete TR, FD, and intrinsic results appear in Appendices C.7, D.8, and E.4, respectively, and implementation details in Appendix F.

Table 3 summarizes performance through task-level ranks. Ranks are computed from the mean primary metric across seeds and exclude task-specific baselines. For each intrinsic task, an encoder is trained separately on the observability-company (Obs) and Uber corpora and evaluated on both, giving four train–test source pairs. LB and EP each average ranks across these pairs. TR averages MAP@10 ranks across the three taxonomies, while FD averages AUPRC ranks across the four post-deployment windows. Intrinsic and extrinsic ranks average their respective tasks, and the last "Overall" column averages these two aggregates.

DAGNN ranks first overall and Qwen3-4B second, but no encoder family dominates every task. DAG-aware encoders lead EP and TR. General graph and DAG-aware encoders share the lead on LB, while a general graph encoder leads FD and a set-based encoder ranks second.

5.1 Intrinsic Tasks

Evaluations are in-source when training and test traces come from the same corpus, and cross-source otherwise. Figure 3 compares the leading encoder from each family across the four source pairs. For EP, DAGNN ranks first on every source pair and PACE second overall. Both are DAG-aware and outperform every set-based and general graph encoder on all four source pairs. EP labels depend on whether error spans form a contiguous parent–child chain. DAGNN's topologically ordered message

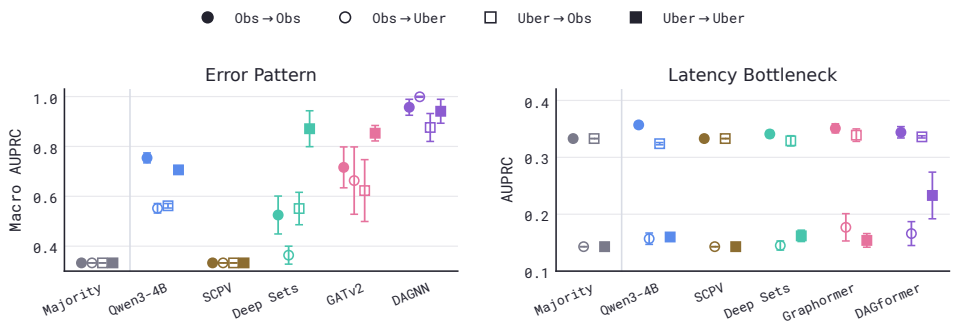


Figure 3: **Intrinsic-task results across trace sources.** Each panel compares Majority with the best encoder in each category. Points and error bars give mean Macro AUPRC (EP) and AUPRC (LB) ± sample standard deviation over five seeds. Colors correspond to input formats (cf. Table 3).

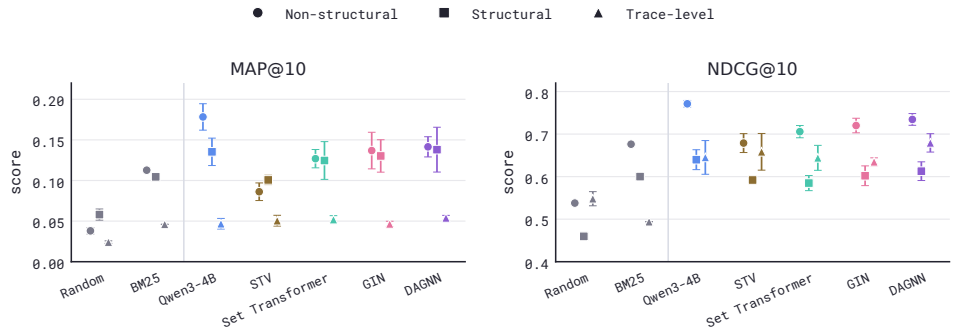


Figure 4: **Trace Retrieval across query taxonomies** ($B = 1$)**.** Points and error bars report mean ± sample standard deviation over five seeds. Each family uses the encoder with the lowest average MAP@10 rank across taxonomies in both panels. Colors correspond to input formats (cf. Table 3).

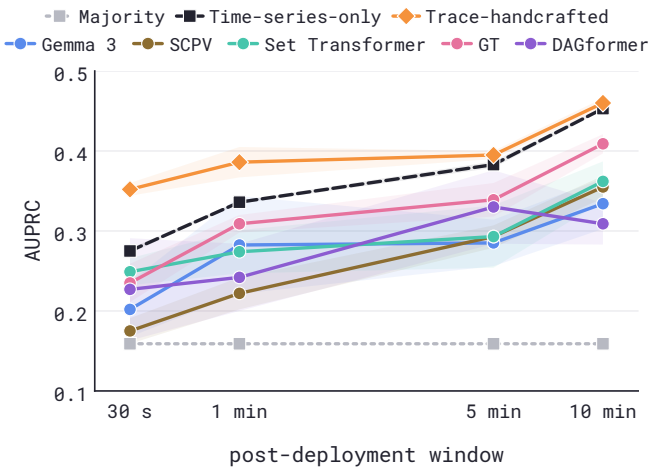
propagation directly follows this partial order, making its inductive bias particularly well aligned with the task. Their consistent advantage over encoders with less specific structural access suggests that DAG-specific inductive bias may benefit EP. LB, although also structural, additionally requires identifying the critical path and comparing exclusive durations globally, for which the advantage of DAG-aware propagation is less direct. LB instead reveals a gap between in-source performance and cross-source generalization. DAGNN leads both in-source settings, whereas Graphormer leads cross-source but improves over the class-prior baselines by only $+0.006$ and $+0.034$ AUPRC.

5.2 Trace Retrieval Across Query Taxonomies

We include two task-specific baselines: Random assigns independent uniform scores to candidate query–trace pairs, while BM25 (Robertson et al., 1994; Robertson & Zaragoza, 2009) is a standard lexical IR baseline that remains competitive across heterogeneous datasets (Thakur et al., 2021). Figure 4 compares the strongest encoder in each family. Performance varies by taxonomy. Language encoders lead on non-structural queries: Gemma 3 reaches 0.181 MAP@10 (Appendix C.7) and Qwen3-4B 0.178, against 0.141 for DAGNN and 0.113 for BM25. On structural queries, DAGNN leads at 0.138, followed by Qwen3-4B at 0.135. GIN and Set Transformer also exceed BM25, but STV does not. Trace-level MAP@10 is lower, with DAGNN at 0.054 and Set Transformer and STV close behind at 0.052 and 0.051. NDCG@10 gives a related but not identical pattern, with Qwen3-4B leading on non-structural and structural queries and DAGNN on trace-level ones.

5.3 Faulty Deployment Across Observation Windows

In Figure 5 (left) we evaluate the five learned trace representation families from Section 4.2 on faulty deployment, combining the embeddings with time-series in the manner described in Section 4.1. However, we find that they all consistently underperform using the time-series alone, leading us to design the handcrafted trace representation described in Section 4.2 and the tail-exceedance aggregation from Section 4.1. This latter approach *does* improve over the time-series baseline at every



Aggregation	Tail	Mean	Attn.
Time-series-only	0.36	**0.36**	**0.36**
Trace-handcrafted	**0.40**	0.32	0.25
Qwen3-4B	0.26	0.28	0.32
Gemma 3	0.28	0.26	0.31
MiniLM	0.27	0.19	0.31
SCPV	0.26	0.16	0.18
STV	0.26	0.16	0.17
Deep Sets	0.28	0.33	0.31
Set Transformer	0.30	0.35	0.31
GIN	0.28	0.30	0.30
GCN	0.22	0.26	0.23
GATv2	0.23	0.27	0.26
Graphormer	0.18	0.16	0.19
SAN	0.18	0.20	0.20
GT	0.32	0.22	0.21
DAGNN	0.24	0.26	0.26
PACE	0.22	0.19	0.18
DAGformer	0.28	0.21	0.29

Figure 5: **FD results** for the top-performing encoders (left) from each family, selected by mean AUPRC across windows, along with three baselines: majority, time-series only (retains only the time-series branch of the three-cell architecture in Section 4.1), and trace-handcrafted (the 25-dimensional encoder described in Section 4.2 and specified in Appendix D). Detailed results by aggregate type (tail-exceedance, mean-pooling, and learned attention) are reported in the table (right), with colors corresponding to input formats (cf. Table 3). Detailed results and standard deviations can be found in Appendix D.8.

horizon, with the largest gain on the most difficult (short-horizon) tasks: $+0.077$ AUPRC at 30 seconds compared with $+0.007$ at 10 minutes. Our finding is thus that traces do contain useful information for the FD task, but modern graph-based encoders do not seem to successfully extract it. To verify that this result is not due to the aggregation method, we evaluate three different pooling approaches for all the encoder varieties, with the results in the table in Figure 5 (right) comparing tail-exceedance with mean-pooling and learned attention (for the latter two we subtract the baseline-pre from each post-deployment summary and concatenate the difference). Aggregation choice does affect the learned encoders, with language encoders performing best under learned attention and both set-based encoders best under mean-pooling, but only the handcrafted approach with tail-exceedance pooling surpasses time-series-only among all evaluated encoder-aggregation pairs.

Cost concerns and frontier LLMs. An important consideration with any trace representation is cost, and this is especially true for faulty deployment, where an assessment must be generated quickly from just-generated trace information. As shown in Table 7, while the call-path and graph encoders are reasonably efficient, the language encoders can take as long as five minutes, precluding usefulness on short-time-horizon FD. Note that a similar issue arises for frontier LLMs, which can also be in-principle applied to FD. While our benchmark's focus is trace *representation*, we do evaluate Claude Opus 5 and GPT-5.6 Sol (both in-context and via an agent harness) on the FD tasks, with results in Appendix D.10. We find that both models do better than the learned embeddings but worse than our handcrafted representation, and that their main strength seems to be in their better use of the time-series modality: when given traces alone, they perform poorly. Furthermore, their latency is between 12 and 73 seconds on average, limiting their practical utility in this time-sensitive application.

6 Conclusion

Iditarod makes trace representation quality measurable through four tasks that examine both individual executions and operational workflows on real-world data. Our evaluation identifies complementary strengths in DAG-aware and language encoders, alongside open challenges in cross-source latency-bottleneck and deployment-level prediction. For the latter, only the handcrafted representation improves on using time-series alone, revealing useful trace signal that none of the sixteen cross-task encoders recovers under the evaluated aggregation schemes. Together, these findings give future pretrained trace encoders concrete targets: preserve execution properties across sources and make them useful across operational tasks. Iditarod supports the development of such encoders, while offering a real-world benchmark of temporally attributed trees for graph foundation model research.

AI USE STATEMENT

The structured queries underlying Iditarod-TR are drawn from the observability company's routine use of its own platform (Section 3.1) and are not AI-generated. We used LLM only to render them, after anonymization, into their natural-language form (Appendix C.4). Generative AI tools also assisted with writing and debugging code. We did not use generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, clean or reformat the trace corpora. Generative AI tools were used to revise text for clarity. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The newly released Iditarod corpus is derived from the observability company's own telemetry, produced by its monitoring of its internal systems, and contains no customer data. Span metadata are anonymized (Appendix A.2). The released fields describe request structure and timing and carry no personal data. The Iditarod corpus and evaluation code are available under the Apache 2.0 License, with the release procedure documented in Appendix A.

REPRODUCIBILITY STATEMENT

The Iditarod data and evaluation code are available under the Apache 2.0 License. Appendix A documents the released span schema, the anonymization procedure, and the release protocol for the corpus. The label constructions are specified in full: budgeted query relaxation and the catalog of atomic relaxation actions in Section 3.1 and Appendix C.3, the latency-bottleneck and error-pattern definitions in Appendix E, and the faulty-deployment labeling criteria in Section 3.2 Train and test partitions are described in Appendix C.6, D.5, E.1, and are released with the data. Appendix F reports the trace input views, encoder configurations and parameter counts, and the task-level optimization settings.

REFERENCES

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A Neural Network Approach to Fast Graph Similarity Computation, March 2020.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset, October 2018.

Maya Bechler-Speicher, Ben Finkelshtein, Fabrizio Frasca, Luis Müller, Jan Tönshoff, Antoine Siraudin, Viktor Zaverkin, Michael M. Bronstein, Mathias Niepert, Bryan Perozzi, Mikhail Galkin, and Christopher Morris. Position: Graph Learning Will Lose Relevance Due To Poor Benchmarks. In *Forty-Second International Conference on Machine Learning Position Paper Track*, June 2025.

Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks? In *International Conference on Learning Representations*, 2022.

Olivier Chapelle and Yi Chang. Yahoo! Learning to Rank Challenge Overview. In *Proceedings of the Learning to Rank Challenge*, pp. 1–24. PMLR, January 2011.

Shuai Ding, Yuepeng E, Jiye Zhang, Liangxiong Li, Lei Zhang, and Jingguo Ge. Trace Anomaly Detection for Microservice Systems via Graph-based Semi-supervised Learning. In *2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 2375–2380, 2024. doi: 10.1109/CSCWD61410.2024.10580078. URL `https://ieeexplore.ieee.org/document/10580078`.

Zehao Dong, Muhan Zhang, Fuhai Li, and Yixin Chen. PACE: A Parallelizable Computation Encoder for Directed Acyclic Graphs. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 5360–5377. PMLR, June 2022.

Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: Yesterday, today, and tomorrow, April 2017.

Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs, January 2021.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long Range Graph Benchmark, November 2023.

Carl Edwards, ChengXiang Zhai, and Heng Ji. Text2Mol: Cross-Modal Molecule Retrieval with Natural Language Queries. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 595–607, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.47.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran Gv, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Harries, Loïc Magne, Isabelle Mohr, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Suppa, Crystina Zhang, Roberta Rocca, Mohammed Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Mariya Hendriksen, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lu, Jordan Clive, Gayatri K, Maksimova Anna, Silvan Wehrli, Maria Tikhonova, Henil Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James V. Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi, Wen-Ding Li, Alessia Borghini, Federico Cassano, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff. MMTEB: Massive Multilingual Text Embedding Benchmark. *International Conference on Learning Representations*, 2025:101715–101771, May 2025.

Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 3–18, Providence RI USA, April 2019. ACM. ISBN 978-1-4503-6240-5. doi: 10.1145/3297858.3304013.

Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, pp. 126–141, New York, NY, USA, November 2022. Association for Computing Machinery. ISBN 978-1-4503-9414-7. doi: 10.1145/3542929.3563482.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272, Sydney, NSW, Australia, August 2017. JMLR.org.

Faegheh Hasibi, Fedor Nikolaev, Chenyan Xiong, Krisztian Balog, Svein Erik Bratsberg, Alexander Kotov, and Jamie Callan. DBpedia-Entity v2: A Test Collection for Entity Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pp. 1265–1268, New York, NY, USA, August 2017. Association for Computing Machinery. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080751.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133. Curran Associates, Inc., 2020.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal Graph Benchmark for Machine Learning on Temporal Graphs. *Advances in Neural Information Processing Systems*, 36:2056–2073, December 2023.

Eeshaan Jain, Indradyumna Roy, Saswat Meher, Soumen Chakrabarti, and Abir De. Graph Edit Distance Evaluation Datasets: Pitfalls and Mitigation. In *The Third Learning on Graphs Conference*, 2024.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, February 2017.

Herb Krasner. The cost of poor quality software in the US: A 2018 report. Technical report, Consortium for IT Software Quality (CISQ), 2018. URL https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking Graph Transformers with Spectral Attention. In *Advances in Neural Information Processing Systems*, volume 34, pp. 21618–21629. Curran Associates, Inc., 2021.

I-Ting Angelina Lee, Zhizhou Zhang, Abhishek Parwal, and Milind Chabbi. The Tale of Errors in Microservices. *Proc. ACM Meas. Anal. Comput. Syst.*, 8(3):46:1–46:36, December 2024. doi: 10.1145/3700436.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 3744–3753. PMLR, May 2019.

Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. Enjoy your observability: An industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27(1):25, 2022a. ISSN 1382-3256. doi: 10.1007/s10664-021-10063-9.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects, May 2019.

Ze Li, Qian Cheng, Ken Hsieh, Yingnong Dang, Peng Huang, Pankaj Singh, Xinsheng Yang, Qingwei Lin, Youjiang Wu, Sebastien Levy, and Murali Chintalapati. Gandalf: An Intelligent, End-To-End Analytics Service for Safe Deployment in Large-Scale Cloud Infrastructure. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 389–402, 2020. ISBN 978-1-939133-13-7.

Zeyan Li, Nengwen Zhao, Shenglin Zhang, Yongqian Sun, Pengfei Chen, Xidao Wen, Minghua Ma, and Dan Pei. Constructing Large-Scale Real-World Benchmark Datasets for AIOps, August 2022b.

Yongjiang Liang and Peixiang Zhao. Similarity Search in Graph Databases: A Multi-Layered Indexing Approach. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 783–794, San Diego, CA, USA, April 2017. IEEE. ISBN 978-1-5090-6543-1. doi: 10.1109/ICDE.2017.129.

Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 48–58. IEEE, 2020. ISBN 978-1-7281-9870-5. doi: 10.1109/ISSRE5003.2020.00014. URL https://ieeexplore.ieee.org/document/9251058/.

Yuankai Luo, Veronika Thost, and Lei Shi. Transformers over Directed Acyclic Graphs. *Advances in Neural Information Processing Systems*, 36:47764–47782, December 2023.

Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data, February 2025.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 Datasets, June 2013.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Samuel Rey, Hamed Ajorlou, and Gonzalo Mateos. Directed Acyclic Graph Convolutional Networks. *IEEE Transactions on Signal Processing*, 74:1847–1862, 2026. ISSN 1941-0476. doi: 10.1109/TSP.2026.3687632.

S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Text Retrieval Conference*, 1994.

Stephen Robertson and Hugo Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019.

Henrique Schechter Vera, Sahil Dua, et al. EmbeddingGemma: Powerful and lightweight text representations. 2025. URL https://arxiv.org/abs/2509.20354.

Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc., 2010. URL http://research.google.com/archive/papers/dapper-2010-1.pdf.

Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software*, 146: 215–232, December 2018. ISSN 01641212. doi: 10.1016/j.jss.2018.09.082.

Yongqian Sun, Minyi Shao, Xiaohui Nie, Kaiwen Yang, Xingda Li, Bowen Hao, Shenglin Zhang, Changhua Pei, Dongbiao He, Yanbiao Li, and Dan Pei. A Comprehensive Benchmark and Empirical Study of Trace Anomaly Detection. *IEEE Transactions on Services Computing*, 18(6): 3364–3377, November 2025. ISSN 1939-1374, 2372-0204. doi: 10.1109/TSC.2025.3622122.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 1, December 2021.

Veronika Thost and Jie Chen. Directed Acyclic Graph Neural Networks. In *International Conference on Learning Representations*, October 2021.

Ellen Voorhees. Overview of the TREC 2004 Robust Track. January 2004.

Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection, May 2020.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MINILM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pp. 5776–5788, Red Hook, NY, USA, December 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6.

Zhe Xie, Changhua Pei, Wanxue Li, Huai Jiang, Liangfei Su, Jianhui Li, Gaogang Xie, and Dan Pei. From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1739–1749. ACM, 2023a. ISBN 979-8-4007-0327-0. doi: 10.1145/3611643.3613861. URL https://dl.acm.org/doi/10.1145/3611643.3613861.

Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. In *Proceedings of the ACM Web Conference 2023*, pp. 2874–2884. ACM, 2023b. ISBN 978-1-4503-9416-1. doi: 10.1145/3543507.3583215. URL https://dl.acm.org/doi/10.1145/3543507.3583215.

Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. Openrca: Can large language models locate the root cause of software failures? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M4qNIzQYpd.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do Transformers Really Perform Badly for Graph Representation? In *Advances in Neural Information Processing Systems*, volume 34, pp. 28877–28888. Curran Associates, Inc., 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Chenxi Zhang, Xin Peng, Tong Zhou, Chaofeng Sha, Zhenghui Yan, Yiru Chen, and Hong Yang. TraceCRL: Contrastive representation learning for microservice trace analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1221–1232. ACM, 2022a. ISBN 9781450394130. doi: 10.1145/3540250.3549146. URL https://dl.acm.org/doi/10.1145/3540250.3549146.

Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-VAE: A Variational Autoencoder for Directed Acyclic Graphs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Shenglin Zhang, Zhongjie Pan, Heng Liu, Pengxiang Jin, Yongqian Sun, Qianyu Ouyang, Jiaju Wang, Xueying Jia, Yuzhi Zhang, Hui Yang, Yongqiang Zou, and Dan Pei. Efficient and Robust Trace Anomaly Detection for Large-Scale Microservice Systems. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 69–79, Florence, Italy, October 2023. IEEE. ISBN 979-8-3503-1594-3. doi: 10.1109/ISSRE59848.2023.00012.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models, June 2025.

Zhen Zhang, Jiajun Bu, Martin Ester, Zhao Li, Chengwei Yao, Zhi Yu, and Can Wang. H2MN: Graph Similarity Learning with Hierarchical Hypergraph Matching Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pp. 2274–2284, New York, NY, USA, August 2021. Association for Computing Machinery. ISBN 978-1-4503-8332-5. doi: 10.1145/3447548.3467328.

Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. {CRISP}: Critical Path Analysis of {Large-Scale} Microservice Architectures. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 655–672, 2022b.

Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Transactions on Software Engineering*, 47(2):243–260, February 2021. ISSN 0098-5589, 1939-3520, 2326-3881. doi: 10.1109/TSE.2018.2887384.

A Shared Data Details & Release Protocol

A.1 Data Schema

Iditarod uses the shared span schema in Table 4 across its four tasks. Task-specific releases pair these trace records with the queries, labels, or time series required by each task.

Table 4: Released span schema. All denotes fields shared by the four tasks.

Field	Task	Definition
`trace_id`	All	Identifier shared by all spans in one trace.
`span_id`	All	Identifier of the span.
`parent_id`	All	Identifier of the parent span, or `0` for the root.
`start_time`	All	Span start time in nanoseconds.
`duration`	All	Span duration in nanoseconds.
`service`	All	Service that emitted the span.
`operation_name`	All	Instrumented operation represented by the span.
`resource_name`	All	Resource or action handled by the operation.
`status`	All	Execution outcome of the span.
`span_kind`	All	Role of the span in a distributed call.
`type`	All	Broad workload category reported by the tracer.
`language`	All	Runtime language reported by the tracer.
`version`	FD	Version of the service that emitted the span.
`instance_id`	FD	Deployment-instance key joining traces, labels, and time series.

Table 5 reports the observed categorical vocabularies. Status is retained unchanged. Span kind, type, and language are replaced with opaque tokens under the anonymization procedure in Appendix A.2, without disclosing their correspondence to the original values.

Table 5: Categorical vocabularies observed before anonymization.

Field	Observed values
`status`	`ok`, `error`
`span_kind`	`client`, `consumer`, `internal`, `producer`, `server`
`type`	`android`, `auth`, `benchmark`, `browser`, `cache`, `cassandra`, `cli`, `cuda`, `custom`, `db`, `elasticsearch`, `gpu`, `grpc`, `http`, `ios`, `llm`, `mongodb`, `python`, `queue`, `redis`, `rpc`, `spark`, `sql`, `sublayer`, `synthetic`, `system`, `template`, `web`, `websocket`, `worker`
`language`	`C++`, `CPython`, `dotnet`, `go`, `java`, `javascript`, `jvm`, `node`, `php`, `python`, `ruby`, `rust`

Trace Retrieval artifacts. The query table contains `query_id`, the natural-language query `nlq`, and its `taxonomy`. For each released budget $B \in \{1, 3\}$, train and test qrels contain `query_id`, `trace_id`, and an integer `relevance` $\in \{0, 1, 2, 3\}$, where larger values indicate greater relevance.

Faulty Deployment artifacts. Each observation horizon contains label, trace, and time-series tables joined by `instance_id`. The label table contains the service, operation, and resource triplet, `target_version`, deployment time `t0`, and the binary `label`. The time-series table provides aligned hit, error, and 50th, 95th, and 99th latency-percentile series for the incumbent and target versions.

Intrinsic-task artifacts. LB and EP each provide source-specific trace tables and train/test label tables for the observability-company and Uber corpora. Both use the shared span schema together with their task-specific labels.

A.2 Anonymization

The Uber corpus is provided in anonymized form (Lee et al., 2024). Traces from the observability company follow a similar de-identification policy across all four tasks. The fields `service`, `resource_name`, `operation_name`, `version`, `span_kind`, `type`, and `language` are replaced by field-specific opaque tokens such as `Service17`. Trace and span identifiers are pseudonymized consistently so that trace membership and parent-child links remain recoverable from the released fields.

For TR and the intrinsic tasks, span start times are expressed relative to the first span of each trace. For FD, a single global offset is subtracted from every deployment, span, and time-series timestamp in the dataset. This removes absolute calendar time while preserving time differences and the global temporal ordering of all FD observations. Durations, relative ordering, and span status are retained, while direct identifiers and absolute timestamps are not released.

Structured trace queries are transformed with the same categorical maps before NLQ generation and release. Their values therefore remain aligned with the pseudonymized TR corpus.

A.3 Global Statistics

Table 6 separates the observability-company corpus from the Uber trace corpus (Lee et al., 2024). Trace and span counts are deduplicated across tasks by `trace_id` and (`trace_id`, `span_id`), respectively. Service identifiers are deduplicated within each source and treated as source-specific when the two corpora are combined.

Table 6: Global graph statistics for the four-task benchmark. Counts are deduplicated across tasks within each source corpus.

Corpus	#Traces	#Spans	#Edges	#Services
Observability Company	3,182,575	236,510,733	233,328,158	2,783
Uber corpus	24,893	4,323,703	4,298,810	584
Full benchmark	3,207,468	240,834,436	237,626,968	3,367

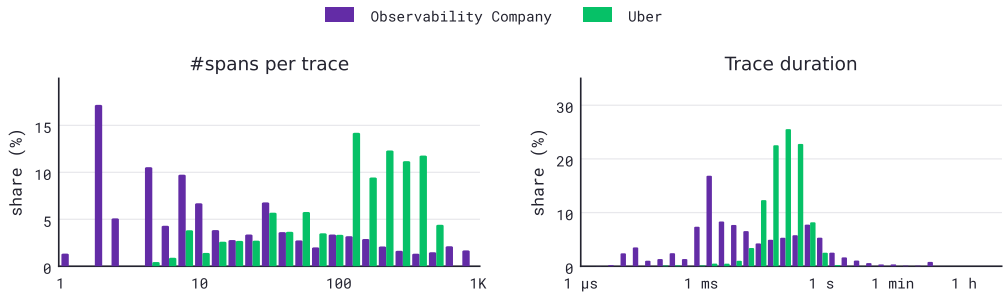


Figure 6: Trace distributions by source across the full benchmark. Bars show per-corpus shares in logarithmically spaced bins.

B Extended Results

B.1 Overall Performance Profile

Figure 7 summarizes performance across Iditarod's 15 benchmark cases: three TR taxonomies, four FD observation windows, and four train–test source pairs for each intrinsic task. Each family is represented by the encoder with the lowest overall average rank in Table 3. For encoder m and benchmark case c, let $s_{m,c}$ denote its score and $s_c^\star = \max_m s_{m,c}$ the best score among the displayed encoders. The performance ratio is $r_{m,c} = s_c^\star / s_{m,c}$. Its profile

$$\rho_m(\tau) = \frac{1}{15} \sum_{c=1}^{15} \mathbb{1}[r_{m,c} \le \tau] \tag{3}$$

is the fraction of cases on which encoder m lies within a factor τ of the best displayed score. Thus, $\rho_m(1)$ is the fraction on which m achieves the best score, while $\rho_m(1.25)$ is the fraction on which it achieves at least 80% of the best score. Curves closer to the upper-left corner indicate stronger and more consistent performance.

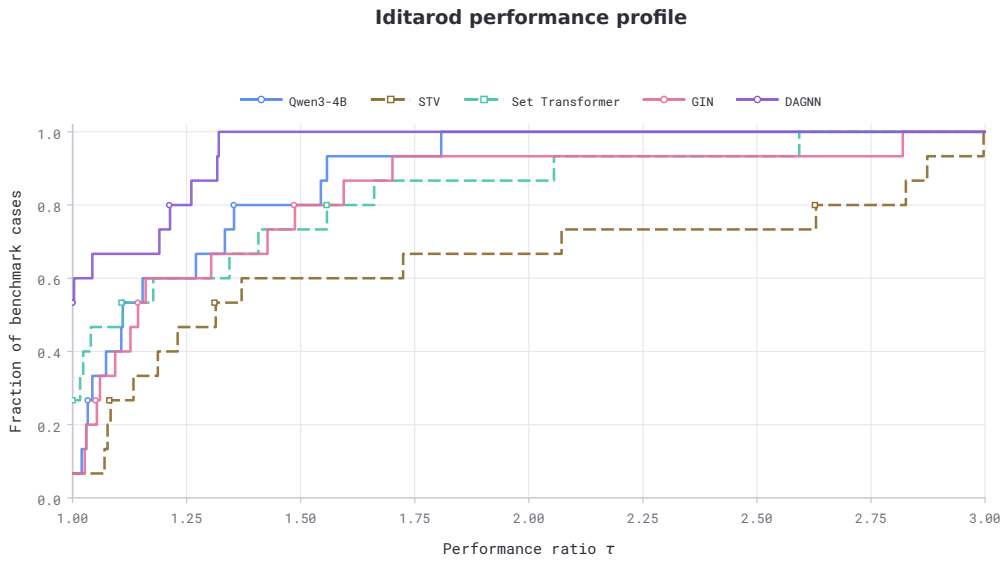


Figure 7: **Iditarod performance profile.** Fraction of benchmark cases for which each family representative lies within a factor τ of the best displayed score. Higher and further left is better.

Among these representatives, DAGNN provides the most consistent performance across the benchmark. It achieves the best score on 8 of the 15 cases, lies within a factor of 1.25 on 12, and is never farther than a factor of 1.32 from the best. Set Transformer leads four cases, while Qwen3-4B, GIN, and STV each lead one. Although no encoder leads every case, DAGNN offers the strongest overall profile among the selected family representatives.

B.2 Model Size–Performance Trade-off and Computational Cost

Figure 8 relates task performance to encoder parameter count for the 14 parameterized encoders. The plotted means span the same benchmark cases as the task-level leaderboard ranks but aggregate raw scores rather than per-case ranks: TR averages MAP@10 across the three query taxonomies, FD averages AUPRC across the four post-deployment windows, and EP and LB average their primary metrics across the four train–test source pairs. STV and SCPV are parameter-free and omitted from the logarithmic parameter axis.

The dashed line marks the Pareto frontier, comprising encoders for which no other is both no larger and no worse.



Figure 8: **Task performance against encoder size.** Scores average raw metrics across the same benchmark cases used for each task-level rank in Table 3. Dashed lines mark Pareto frontiers. Colors correspond to input formats (cf. Table 3).

Inference latency. Table 7 reports mean inference latency across Iditarod's four tasks, measured individually over each test set using an eight-GPU NVIDIA A100 setup. The timed input is a deployment instance for FD, a query–trace pair for TR, and an individual trace for EP and LB.

Call-path encoders are the fastest on TR, EP, and LB. On FD, all evaluated non-language encoders process an instance in under one second on average, whereas language encoders require seconds to minutes. Language encoders process serialized traces in multiple chunks (Table 29, Appendix F), adding computational work beyond what parameter count alone reflects. These measurements complement the performance results by quantifying inference cost under our evaluation setup.

Table 7: **Encoder size and inference latency.** Latencies are means per task input. Parameter counts cover only the encoder, excluding task-specific modules. Colors correspond to input formats (cf. Table 3).

Encoder	# encoder parameters	FD / instance	TR / query–trace	EP / trace	LB / trace
Qwen3-4B	4B	274.80 s	3.20 s	1.39 s	743.40 ms
Gemma 3	300M	47.33 s	361.51 ms	259.25 ms	148.75 ms
MiniLM	22.7M	5.68 s	73.02 ms	36.89 ms	22.03 ms
SCPV	0	14.08 ms	0.83 ms	0.26 ms	0.25 ms
STV	0	21.10 ms	0.83 ms	0.28 ms	0.25 ms
Deep Sets	1.48M	13.46 ms	2.89 ms	1.87 ms	1.83 ms
Set Transformer	1.45M	42.75 ms	12.66 ms	10.66 ms	10.60 ms
GIN	0.80M	17.39 ms	6.74 ms	3.37 ms	3.33 ms
GCN	0.60M	17.82 ms	4.31 ms	3.49 ms	3.45 ms
GATv2	1.07M	24.12 ms	5.82 ms	4.93 ms	4.85 ms
Graphormer	43.8M	751.80 ms	14.66 ms	13.64 ms	13.40 ms
SAN	2.17M	897.28 ms	35.28 ms	30.57 ms	32.76 ms
GT	1.84M	51.69 ms	16.81 ms	14.04 ms	14.08 ms
DAGNN	0.97M	92.95 ms	43.30 ms	31.32 ms	24.72 ms
PACE	1.07M	296.48 ms	6.85 ms	5.94 ms	5.77 ms
DAGformer	1.71M	678.25 ms	14.63 ms	13.25 ms	13.28 ms

B.3 Cross-Task Alignment

Table 8 compares task-level encoder ranks using Spearman correlation. LB and EP have the most similar orderings ($\rho = 0.644$), while the intrinsic aggregate has a modest positive association with TR ($\rho = 0.356$) but no comparable positive association with FD ($\rho = -0.250$).

This difference is consistent with where each task places its supervision. LB and EP label individual traces directly. TR is extrinsic and cross-modal, but its relevance label applies to a query–trace pair and is determined by properties observable in the trace. In FD, the label applies to a deployment instance rather than to any individual trace. Useful trace evidence may affect only a subset of requests and must survive within-cell pooling, cross-cell comparison, and fusion with time-series evidence.

Table 8: Spearman correlations between task-level rank vectors, with the intrinsic aggregate compared against the extrinsic tasks.

	LB	EP	TR	FD
LB	+1.000	+.644	+.233	-.223
EP	+.644	+1.000	+.392	-.274
TR	+.233	+.392	+1.000	-.021
FD	-.223	-.274	-.021	+1.000
Intrinsic avg.	–	–	+.356	-.250

C Trace Retrieval Details

C.1 Positioning with respect to information retrieval benchmarks

Shared evaluation suites such as BEIR (Thakur et al., 2021) and MMTEB (Enevoldsen et al., 2025) have been instrumental in driving progress in text retrieval, enabling systematic comparison of embedding models across diverse domains. However, as shown in Table 9, the vast majority of IR datasets operate on unstructured text, and even learning-to-rank benchmarks that support graded relevance typically release pre-computed numeric features rather than raw documents (Qin & Liu, 2013; Chapelle & Chang, 2011). Beyond text, graph-based retrieval has primarily focused on structural similarity search between graphs (Bai et al., 2020; Jain et al., 2024; Liang & Zhao, 2017;

Li et al., 2019; Zhang et al., 2021), while the setting of graph semantic search, retrieving graphs from natural language queries, remains largely underexplored comparatively (Edwards et al., 2021). As with the graph learning benchmarks, observability is notably absent from all these lines of work. This gap is compounded by the cost of relevance labeling, which typically requires expert annotators to judge thousands of query-document pairs (Thakur et al., 2021).

Table 9: Comparison with IR datasets. #Qrels reports the number of annotated query–document pairs. For MS MARCO and Text2Mol, only relevant pairs are recorded; unjudged pairs are assumed non-relevant.

Dataset	Domain	Doc. Type	#Queries	#Qrels	Relevancy
MS MARCO (2018)	Web	Text	1M+	592K+	Binary
Robust04 (2004)	News	Text	249	311K+	3-levels
TREC-COVID (2020)	Bio-Medical	Text	50	69K+	3-levels
DBPedia-Entity (2017)	Wikipedia	Text	467	49K+	3-levels
MQ2007 (2013)	Web	Text	1,692	69K+	3-levels
MSLR-WEB30K (2013)	Web	Text	31,531	3M+	5-levels
Yahoo! (2011)	Web	Text	29,921	710K+	5-levels
Text2Mol (2021)	Chemistry	Graph	33,010	33K+	Binary
Iditarod-TR	**Observability**	**Graph**	**3,980**	**1.1M+**	**4-levels**

C.2 Structured Query Syntax

A structured query contains three components. The `span_queries` map defines aliases whose predicates select spans, the `trace_query` combines these aliases through logical and structural operators, and the optional `trace_level_filters` constrain aggregate trace properties. An alias predicate may combine categorical attributes, numerical comparisons, and one of three scopes. Scopes are expressed as `scope:root`, `scope:entry`, and `scope:any`, which select the trace root, a service-entry span, and any span, respectively. An omitted scope defaults to `scope:any`. Trace-level filters may constrain `trace_duration`, `total_span_count`, or `span_count(a)` for an alias a.

Table 10: Logical and structural operators supported by structured trace queries.

Operator	Semantics
`a => b`	A span matching `a` is the direct parent of a span matching `b`.
`a -> b`	A span matching `a` is an ancestor of a span matching `b`.
`a && b`	Both expressions hold within the trace.
`a || b`	At least one expression holds within the trace.
`NOT(a)`	The enclosed alias or relational expression does not hold.
`(a)`	Parentheses group expressions explicitly.

```
span_queries:
  a: "scope:root service:Service17"
  b: "service:Service31"
  c: "status:error"
trace_query: "a -> b => c"
trace_level_filters: "trace_duration:[1s TO 5s]"
```

```
span_queries:
  a: "scope:entry service:Service42"
  b: "operation_name:OperationName9"
  c: "status:error"
trace_query: "(a -> b) && NOT(c)"
trace_level_filters: "trace_duration:[10s TO 100s]"
```

C.3 Relaxation Policy

Table 11 completes the catalog of atomic transformations introduced in Section 3.1. At step t, let $\mathcal{A}(q^{(t)})$ contain every target-specific transformation whose preconditions are satisfied. The next step

is sampled uniformly from this set,

$$a_t \sim \mathrm{Unif}\Big(\mathcal{A}(q^{(t)})\Big), \qquad q^{(t+1)} = a_t\Big(q^{(t)}\Big). \tag{4}$$

Each selected transformation consumes one unit of relaxation budget. The policy admits only transformations that preserve the expansion property in Section 3.1. In a negated context, this requires strengthening the inner predicate rather than weakening it.

Table 11: Atomic query-relaxation actions.

Component	Action	Transition
Predicate	Drop	Remove one categorical span predicate or trace-level filter.
Predicate	Relax threshold	Decrease a lower bound or increase an upper bound. Reverse these directions inside a negation.
Scope	Widen	Follow ROOT→ENTRY→ANY. Inside a negation, follow the reverse order.
Relation	Weaken	Follow direct parent-child→ancestor-descendant→co-occurrence. Inside a negation, replace ancestor-descendant by direct parent-child.
Alias literal	Drop	Remove one positive or negated occurrence while retaining its span predicate and scope definition.
Alias literal	Remove	Remove the occurrence together with its definition when no remaining constraint depends on it.
Conjunction	Split	Replace a multi-literal conjunction by a disjunction between one selected literal and the conjunction of the remaining literals.
Negated relation	Remove	Remove the relational literal once its inner relation and associated constraints reach the policy boundary.
Trace query	Drop	Replace TQ by TRUE while retaining the trace-level filter TF.

Adaptive numerical steps. For a numerical attribute r, we estimate its relaxation magnitude from the empirical percentiles obtained in the context of the remaining query constraints. Writing $p_u^{(r)}$ for percentile u, the step is the mean spacing between consecutive five-percentile points in the interquartile range,

$$\delta_r = \frac{1}{10} \sum_{u \in \{25,30,\ldots,70\}} \Big(p_{u+5}^{(r)} - p_u^{(r)}\Big). \tag{5}$$

The fixed fallbacks are 1 ms for span duration, 10 ms for trace duration, five spans for total span count, and one span for alias-specific span count.

Released configuration. We sample $N = 5$ paths by selecting uniformly among admissible actions and release the benchmark for $B \in \{1, 3\}$. When a query-trace pair is recovered through several paths, its highest observed relevance is retained.

C.4 NLQ Generation

We generate each natural-language query from the anonymized structured query rather than from a retrieved trace. The released corpus was generated with GPT-4.1 nano.

To diversify surface form, each query receives one of the 15 style instructions and one term from each lexical pool reproduced in Prompt 1. Their combination fills the `style_instruction` field in the generation prompt. The released NLQ table is the artifact of record.

Prompt 1: Style instructions and lexical variations used for NLQ generation.

```
BASE STYLE INSTRUCTIONS

1. Write a neutral declarative sentence describing the topology (which services are upstream/downstream of
     which). Example tone: "Traces where Service209 is upstream of Service312"

2. Write a casual imperative sentence describing the topology. Example tone: "Show me traces with Service209
     above Service312"

3. Write a technical noun phrase describing the topology. Example tone: "Service209-upstream-of-Service312
     traces with errors"

4. Write a neutral declarative sentence framing the query as a request flow between services. Example tone:
     "Traces where requests flow from Service209 to Service312"

5. Write a casual imperative sentence framing the query as a request flow. Example tone: "Give me requests
     going from Service209 to Service312"

6. Write a technical noun phrase framing the query as a request flow. Example tone: "Service209-to-Service312
     request paths, latency >1s"

7. Write a neutral declarative sentence framing the query as causal (one service triggers/spawns activity in
     another). Example tone: "Traces where Service209 triggers spans in Service312"

8. Write a casual question framing the query as causal. Example tone: "Which traces have Service209
     triggering Service312?"

9. Write a technical declarative sentence framing the query as causal. Example tone: "Traces with Service209
     spawning downstream Service312 spans, status:error"

10. Write a neutral declarative sentence that leads with the filter condition (duration, span count, error
     status). Example tone: "Slow traces where Service209 calls Service312"

11. Write a casual imperative sentence that leads with the filter condition. Example tone: "Get me the slow
     calls from Service209 to Service312"

12. Write a technical noun phrase that leads with the filter condition. Example tone: "High-latency
     Service209-to-Service312 traces, >500ms"

13. Write a formal declarative sentence using relative clauses. Example tone: "Traces in which Service209
     invokes Service312"

14. Write a terse noun phrase (under 8 words), no verb needed. Example tone: "Service209 to Service312 errors"

15. Write a casual question in natural conversational English. Example tone: "Any traces with Service209
     hitting Service312?"

LEXICAL VARIATIONS

Retrieval verbs:
show | get | list | return | fetch | retrieve | pull | find | give me | grab | look up | search for | surface
     | extract | display

Trace nouns:
traces | requests | calls | spans | transactions | invocations | executions | operations | flows

Relationship verbs:
calls | invokes | reaches | hits | triggers | spawns | connects to | flows to | leads to | talks to |
     contacts | sends to | forwards to | accesses

COMPOSITION

<base style instruction>
Use the verb "<retrieval verb>", refer to traces as "<trace noun>", and express service relationships with
     "<relationship verb>".
```

Prompt 2: NLQ generation prompt.

```
You are generating a natural-language query (NLQ) for a trace retrieval benchmark.
The goal is to describe WHAT traces to retrieve, not WHY or for what investigation purpose.

You will be given a JSON object with these fields:
- "span_queries": a dict mapping aliases (e.g., "a", "b") to a query string (e.g., "service:Service209
    @env:staging")
- "trace_query": a string combining aliases with operators (e.g., "a", "a -> b", "a => b", "a && b", "a || b")
- "trace_level_filters": either null or a string of extra filters (e.g., "trace_duration:>500ms
    total_span_count:>=20")

### Operator semantics (must be respected)
- "a": traces containing spans matching alias a.
- "a -> b": traces where a span matching a is an ANCESTOR of a span matching b (any depth --- possibly
    through intermediate spans).
- "a => b": traces where a span matching a is the DIRECT PARENT of a span matching b (immediate, no
    intermediaries).
- "a && b": traces containing both patterns anywhere.
- "a || b": traces containing either pattern.

CRITICAL --- "->" vs "=>" distinction:
- For "->": do NOT use "directly" or "immediate" --- the relationship may span multiple levels.
- For "=>": convey that the relationship is immediate (one level). Vary how you express it:
  "directly calls", "is the immediate parent of", "spawns", "hands off to as next hop",
  "forwards without intermediary to", "sits one level above", etc.
- This distinction matters.

### Scope semantics
- "@_trace_root:1": the span is the root of the trace.
- "@_top_level:1": the span is a service-entry span.

### Hard constraints
- Use ONLY information from the input JSON.
- Do not mention alias letters ("a", "b") in output.
- Output exactly one English sentence or phrase.
- Treat anonymized names (Service123, Operationname45) as proper nouns.

### Intent constraint
This is RETRIEVAL only. Describe WHAT to fetch, not WHY.
- GOOD: "Traces from Service209 to Service312"
- BAD: "Debug why Service209 is slow"

### Style
{style_instruction}

Now produce ONE retrieval query following the style above.
```

C.5 Human Validation of Graded Relevance

For each comparison, annotators saw only a natural-language query and two traces and answered, "Which trace is more relevant to the query?" They responded independently, blind to the qrels and to one another's responses. The evaluation contains 360 trace-pair comparisons drawn from 90 queries, with 120 comparisons per query taxonomy and 180 per relaxation budget. The relevance gap covers every unequal label pairing: $\Delta\,\mathrm{rel} = 1$ comprises 3 vs. 2, 2 vs. 1, and 1 vs. 0, $\Delta\,\mathrm{rel} = 2$ comprises 3 vs. 1 and 2 vs. 0, and $\Delta\,\mathrm{rel} = 3$ comprises 3 vs. 0. 5 observability engineers participated.

Human-qrel agreement is the fraction of judgments in which the selected trace has the higher qrel score, averaged across comparisons. Pairwise inter-annotator agreement is the fraction of agreeing annotator pairs, averaged across comparisons with multiple judgments. Figure 9 reports human-qrel agreement by query taxonomy, relaxation budget, and relevance gap.

C.6 Train/Test Splitting

TR assigns query identifiers to an 80/20 train-test partition stratified by query taxonomy. We then make the trace corpora disjoint: if a trace occurs in qrels associated with both partitions, it is assigned to one partition and all its qrel entries are removed from the other. Consequently, neither query nor trace identifiers overlap between train and test.

C.7 Extended Results

Let $x_{q,i}$ denote the trace ranked at position i for query q, and let $r_{q,i} = \mathrm{rel}(q, x_{q,i})$. NDCG@10 evaluates the full graded relevance scale through

$$\mathrm{DCG@10}(q) = \sum_{i=1}^{10} \frac{2^{r_{q,i}} - 1}{\log_2(i+1)}, \qquad \mathrm{NDCG@10}(q) = \frac{\mathrm{DCG@10}(q)}{\mathrm{IDCG@10}(q)}, \tag{6}$$

where IDCG@10 is the maximum DCG@10 attainable for q. We average NDCG@10 over all test queries.

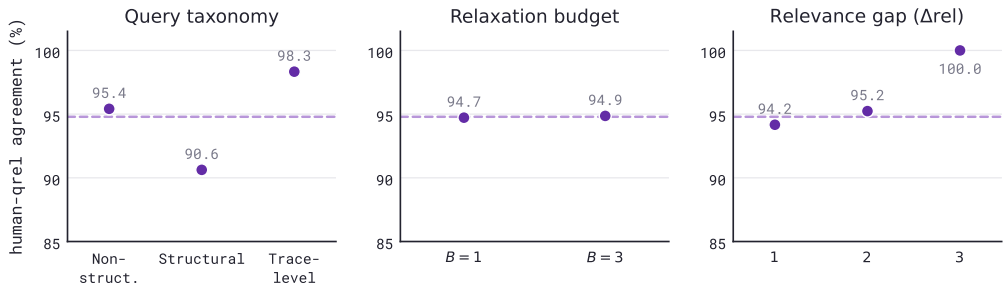


Figure 9: **Human validation of TR relevance labels.** Agreement between human preferences and the qrel ordering by query taxonomy, relaxation budget, and relevance gap. The dashed line marks the 94.8% overall agreement.

MAP@10 is evaluated only on queries with at least one exact match. For these queries, a trace is relevant if and only if $\mathrm{rel}(q, x) = 3$. Define $e_{q,i} = \mathbb{1}[r_{q,i} = 3]$, $P@i(q) = i^{-1} \sum_{j=1}^{i} e_{q,j}$, and let R_q be the total number of exact matches for q. Then

$$\mathrm{AP@10}(q) = \frac{1}{R_q} \sum_{i=1}^{10} P@i(q) e_{q,i}. \tag{7}$$

MAP@10 is the mean of AP@10 over this subset of test queries. Tables 12 and 13 report the full results for the two released relaxation budgets.

Table 12: Iditarod-TR results for $B = 1$. Values are mean $\pm$ sample standard deviation over five seeds. **bold** = best, underline = second-best.

	MAP@10			NDCG@10		
Encoder	Non-struct.	Structural	Trace-level	Non-struct.	Structural	Trace-level
Random	.038 ± .004	.058 ± .007	.024 ± .002	.538 ± .004	.460 ± .009	.548 ± .017
BM25	.113 ± .000	.104 ± .000	.046 ± .000	.676 ± .000	.600 ± .000	.494 ± .000
Qwen3-4B	.178 ± .016	.135 ± .017	.047 ± .007	**.771 ± .008**	.640 ± .023	.645 ± .040
Gemma 3	**.181 ± .009**	.132 ± .007	.038 ± .003	.768 ± .009	**.654 ± .014**	.519 ± .027
MiniLM	.107 ± .006	.093 ± .007	.037 ± .002	.720 ± .010	.602 ± .007	.608 ± .009
SCPV	.101 ± .010	.119 ± .007	.042 ± .004	.706 ± .017	.601 ± .006	.594 ± .032
STV	.086 ± .011	.101 ± .006	.051 ± .007	.679 ± .022	.592 ± .008	.658 ± .043
Deep Sets	.130 ± .018	.124 ± .028	.050 ± .004	.699 ± .011	.590 ± .011	.645 ± .037
Set Transformer	.127 ± .011	.125 ± .023	.052 ± .005	.706 ± .014	.585 ± .018	.644 ± .029
GIN	.137 ± .023	.130 ± .020	.047 ± .003	.720 ± .017	.602 ± .023	.635 ± .009
GCN	.136 ± .022	.128 ± .025	.049 ± .004	.714 ± .021	.609 ± .025	.646 ± .027
GATv2	.106 ± .019	.106 ± .016	.043 ± .010	.680 ± .015	.573 ± .025	.623 ± .077
Graphormer	.114 ± .013	.123 ± .019	.050 ± .006	.694 ± .013	.596 ± .015	.651 ± .044
SAN	.090 ± .031	.085 ± .014	.039 ± .009	.672 ± .051	.587 ± .045	.478 ± .058
GT	.081 ± .023	.103 ± .020	.039 ± .014	.656 ± .042	.584 ± .027	.513 ± .109
DAGNN	.141 ± .012	**.138 ± .028**	**.054 ± .003**	.734 ± .014	.613 ± .022	**.679 ± .022**
PACE	.130 ± .014	.115 ± .024	.052 ± .005	.725 ± .020	.607 ± .027	.675 ± .030
DAGformer	.084 ± .024	.081 ± .015	.036 ± .004	.626 ± .034	.516 ± .026	.465 ± .032

Table 13: Iditarod-TR results for $B = 3$. Values are mean $\pm$ sample standard deviation over five seeds. **bold** = best, underline = second-best.

	MAP@10			NDCG@10		
Encoder	Non-struct.	Structural	Trace-level	Non-struct.	Structural	Trace-level
Random	.047 ± .010	.066 ± .011	.036 ± .003	.623 ± .006	.550 ± .017	.555 ± .017
BM25	.194 ± .000	.147 ± .000	.046 ± .000	.680 ± .000	.601 ± .000	.417 ± .000
Qwen3-4B	.254 ± .026	.211 ± .024	.084 ± .008	**.765 ± .022**	**.712 ± .024**	.732 ± .058
Gemma 3	**.261 ± .030**	.202 ± .020	.077 ± .016	.755 ± .014	.700 ± .024	.693 ± .101
MiniLM	.162 ± .010	.092 ± .017	.065 ± .008	.703 ± .009	.610 ± .014	.673 ± .039
SCPV	.115 ± .019	.133 ± .023	.071 ± .008	.667 ± .021	.647 ± .030	.630 ± .051
STV	.125 ± .013	.127 ± .019	.061 ± .018	.675 ± .012	.635 ± .019	.586 ± .111
Deep Sets	.165 ± .032	.151 ± .035	.086 ± .009	.695 ± .006	.646 ± .018	.751 ± .051
Set Transformer	.155 ± .013	.160 ± .040	.075 ± .011	.694 ± .024	.660 ± .038	.695 ± .061
GIN	.206 ± .016	.178 ± .054	.075 ± .014	.736 ± .018	.680 ± .047	.701 ± .066
GCN	.205 ± .029	.184 ± .038	.082 ± .012	.719 ± .027	.672 ± .026	.727 ± .041
GATv2	.122 ± .031	.155 ± .035	.061 ± .014	.673 ± .012	.636 ± .028	.628 ± .075
Graphormer	.224 ± .039	.181 ± .023	.078 ± .011	.739 ± .019	.684 ± .014	.714 ± .039
SAN	.192 ± .041	.155 ± .052	.041 ± .002	.693 ± .015	.619 ± .049	.439 ± .047
GT	.168 ± .040	.112 ± .028	.043 ± .009	.701 ± .032	.615 ± .031	.468 ± .088
DAGNN	.257 ± .057	**.224 ± .033**	.074 ± .010	.740 ± .027	.687 ± .021	.701 ± .047
PACE	.212 ± .018	.170 ± .027	**.089 ± .018**	.716 ± .009	.681 ± .021	**.762 ± .085**
DAGformer	.151 ± .059	.147 ± .031	.063 ± .018	.675 ± .038	.612 ± .035	.637 ± .133

C.8 Dataset distributions

Table 14 reports the released query, corpus, and relevance-label counts separately for each relaxation budget. The union row deduplicates query and trace identifiers across budgets.

Table 14: Iditarod-TR dataset statistics by relaxation budget. Counts combine the train and test partitions.

B	#Queries	#Traces	#Qrels	rel = 0	rel = 1	rel = 2	rel = 3
1	3,857	24,946	786,778	168,730	287,603	267,379	63,066
3	2,264	20,131	347,843	11,795	43,600	241,217	51,231
Union	3,980	45,077	1,134,621	180,525	331,203	508,596	114,297

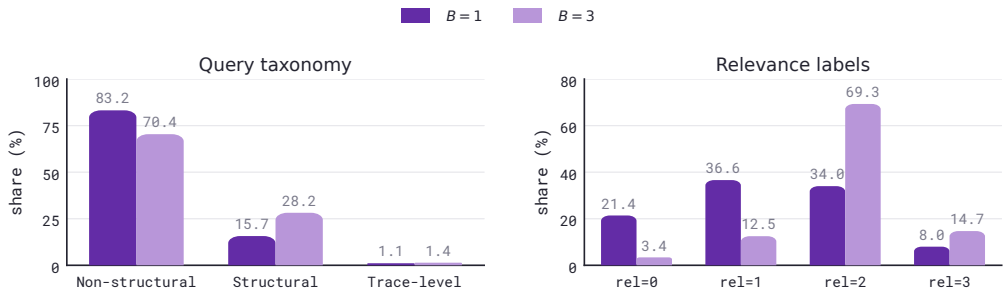


Figure 10: Iditarod-TR distributions by relaxation budget. Left: fraction of unique queries in each taxonomy. Right: fraction of query-trace pairs at each relevance level. Percentages are normalized independently within each budget, and relevance increases from rel $= 0$ to rel $= 3$.

Representation-size calculation. From a sample of 24,946 traces, the serialized JSON representations occupy 1.17 GB in total, or 46.99 kB per trace on average. A Snappy-compressed Parquet encoding of the same fields occupies 304.27 MB in total, or 12.20 kB per trace. A 512-dimensional float32 embedding requires 2.048 kB per trace. The JSON-to-embedding ratio is $46.99/2.048 = 22.94$, reported as $23\times$ in Section 3.1, while the Parquet-to-embedding ratio is $12.20/2.048 = 5.96$.

D Faulty Deployment Details

D.1 Benchmark Comparisons

Table 15: Comparison with anomaly detection benchmarks. Service counts are comma-separated per system.

Benchmark	Modality	Environment	Faults	#Services
RCAEval (2025)	traces, time series, & logs	Testbed	Injected	12, 15, 64
TADBench (2025)	traces	Mixed	Injected	29, 10, 10, 40, 37
OpenRCA (2025)	traces, time series, & logs	Real	Injected	31, 6, 10
Iditarod-FD	traces & time series	Real	Real	2.3K

D.2 Instance Construction

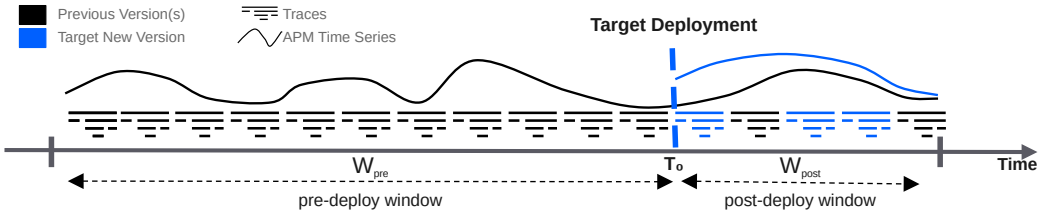


Figure 11: **Faulty Deployment instance construction.** Traces and APM time series are aligned at deployment time T_0 and partitioned by time and version into baseline-pre, baseline-post, and new-post cells.

D.3 Three-cell FD architecture

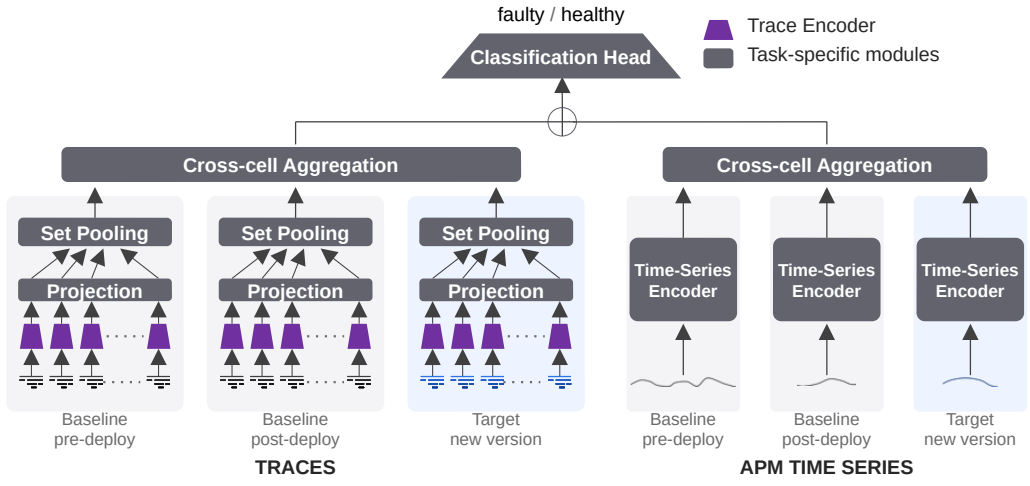


Figure 12: **Three-cell FD architecture.** Per-trace encoding, within-cell set pooling, and cross-cell aggregation produce h_T, which is combined with the APM time-series block h_{TS} for classification.

D.4 Tail-Exceedance Aggregation

Numerical configuration. We set $\alpha_\ell = 0.05$ and $\alpha_u = 0.95$, yielding the 5th and 95th empirical quantiles of baseline-pre as the lower and upper reference thresholds. The baseline-pre cell is pooled into these two per-dimension quantiles and a reference scale. Each post-deployment cell is instead represented by its $K = 64$ largest and $K = 64$ smallest values per dimension. The lower- and upper-tail rates are then averaged.

The denominator of each rate is the full cell size N_c, and comparisons use strict inequalities. The baseline-pre cell provides the thresholds but contributes no separate block to h_T.

Straight-through gradients. Hard indicators prevent gradients from reaching a trainable trace encoder. Let $\Delta^-_{c,n,j} = \ell_j - z_{c,n,j}$ and $\Delta^+_{c,n,j} = z_{c,n,j} - u_j$ denote signed distances into the lower and upper tails. For either tail, we use

$$h = \mathbb{1}[\Delta > 0], \qquad s = \mathrm{sigmoid}\left(\frac{\Delta}{\tau \sigma_j}\right), \qquad \widetilde{h} = s + \mathrm{stopgrad}(h - s). \tag{8}$$

Here, $\mathrm{stopgrad}(\cdot)$ is the identity in the forward pass and has zero derivative. Thus $\widetilde{h} = h$ in the forward pass, while its backward derivative is that of s. For trainable encoders, we set $\tau = 0.1$. The reference scale σ_j is the per-dimension standard deviation in baseline-pre, lower-bounded for numerical stability and detached from the gradient graph.

Block dimensions and normalization. In the primary configuration, the shared projection maps every trace to 25 dimensions. Averaging the lower and upper rates leaves 25 coordinates per post-deployment cell, so concatenating new-post and baseline-post produces a 50-dimensional trace block. In the primary multimodal experiments, the time-series branch uses no learned temporal encoder: it flattens and concatenates the aligned per-cell APM series and engineered scalar features, then applies standardization fitted on the training split. The trace and time-series blocks are concatenated before the binary classification head.

D.5 Train/Test Splitting

All four FD subtasks use the same deployment instances and split membership. We reserve the final two weeks of the dataset for testing and enforce disjoint services across train and test. Using the widest observation window, from one hour before to ten minutes after deployment, we exclude any instance whose window crosses the temporal boundary. Shorter-horizon subtasks inherit this partition and differ only in the post-deployment evidence exposed to the model.

D.6 Dataset distributions

All four observation horizons share the same 1,158 labeled deployments, comprising 169 faulty and 989 healthy instances. At the widest 10-minute horizon, the active release contains 3,108,538 unique traces, 231,638,115 unique spans, and 228,529,577 parent-child edges across 2,305 services. The median instance contains 1,022 traces and the 95th percentile contains 10,604.

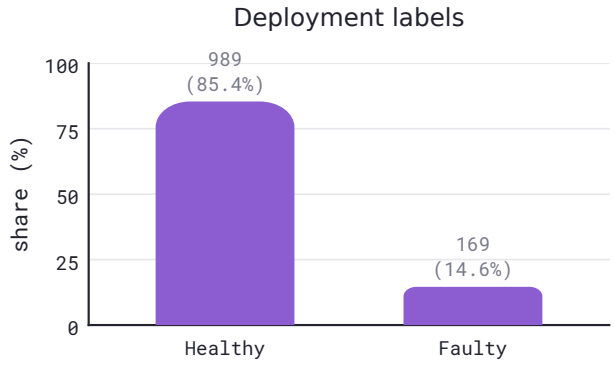


Figure 13: Iditarod-FD label distribution, shared by all four observation horizons.

D.7 Task-Specific Baselines

- **Majority.** Every instance receives the faulty-deployment prevalence measured on the training split.
- **Version-error delta.** Instances are scored by the difference between the new-post and baseline-post overall error rates.

- **Time-series-only.** The trace branch is removed, and the identity time-series encoder passes the three cells' error-rate series to the logistic head.
- **Trace-zero.** The multimodal architecture is retained, but every trace is represented by a 25-dimensional zero vector.
- **Trace-handcrafted.** Let a trace be $x = (V, E)$, with spans V and parent-child edges E. Let $\mathcal{D} = \{0, 1, 2, 3, 4+\}$ denote the depth buckets and $\mathcal{A}$ the anonymized values taken directly by the span attribute `type` defined in Table 4. The representation concatenates

$$d_b(x) = \frac{1}{|V|} \sum_{v \in V} \mathbb{1}[\text{bucket}(\text{depth}(v)) = b], \qquad b \in \mathcal{D},$$

$$p_a(x) = \frac{1}{\max(1, |E|)} \sum_{(u,v) \in E} \mathbb{1}[\text{type}(u) = a], \quad a \in \mathcal{A}.$$

  Their concatenation produces a parameter-free $(|\mathcal{D}| + |\mathcal{A}|)$-dimensional vector.

Trace-zero and Trace-handcrafted use the same Tail-exceedance aggregation and linear logistic head as the evaluated trace encoders.

D.8 Extended Results

Tables 16 and 17 report the full threshold-free results for all 16 trace encoders and the task-specific references used in Section 5.3. As a control, Trace-zero differs from Time-series-only by at most 0.006 AUPRC across observation windows, indicating that an uninformative trace branch leaves time-series-based performance essentially unchanged.

Table 16: FD AUPRC results. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best.

Model	FD-30s	FD-1m	FD-5m	FD-10m
Task-specific references				
Majority	.159 ± .000	.159 ± .000	.159 ± .000	.159 ± .000
Version-error delta	.264 ± .000	.288 ± .000	.328 ± .000	.338 ± .000
Time-series-only	.275 ± .003	.336 ± .002	.383 ± .004	.453 ± .002
Trace + Time Series				
Trace-zero	.281 ± .010	.341 ± .006	.386 ± .002	.451 ± .006
Trace-handcrafted	**.352 ± .008**	**.386 ± .019**	**.395 ± .005**	**.460 ± .006**
Qwen3-4B	.184 ± .029	.247 ± .028	.282 ± .027	.345 ± .012
Gemma 3	.202 ± .034	.282 ± .061	.285 ± .029	.334 ± .032
MiniLM	.230 ± .029	.265 ± .033	.276 ± .028	.322 ± .022
SCPV	.175 ± .016	.222 ± .020	.293 ± .014	.355 ± .014
STV	.203 ± .018	.209 ± .010	.273 ± .009	.343 ± .019
Deep Sets	.218 ± .015	.245 ± .024	.287 ± .008	.364 ± .006
Set Transformer	.249 ± .017	.274 ± .027	.293 ± .039	.362 ± .025
GIN	.243 ± .008	.239 ± .018	.268 ± .025	.371 ± .016
GCN	.176 ± .018	.200 ± .022	.238 ± .049	.271 ± .029
GATv2	.196 ± .019	.209 ± .025	.217 ± .023	.306 ± .040
Graphormer	.188 ± .024	.200 ± .041	.166 ± .013	.169 ± .034
SAN	.189 ± .020	.166 ± .024	.171 ± .041	.191 ± .034
GT	.235 ± .024	.309 ± .011	.339 ± .021	.409 ± .012
DAGNN	.206 ± .029	.230 ± .035	.223 ± .023	.281 ± .023
PACE	.190 ± .043	.192 ± .036	.237 ± .040	.255 ± .063
DAGformer	.227 ± .064	.242 ± .042	.330 ± .046	.309 ± .026

Table 17: FD AUROC results. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best.

Model	FD-30s	FD-1m	FD-5m	FD-10m
Task-specific references				
Majority	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Version-error delta	.651 ± .000	.675 ± .000	.683 ± .000	.699 ± .000
Time-series-only	.667 ± .003	.677 ± .003	.723 ± .014	.738 ± .002
Trace + Time Series				
Trace-zero	**.667 ± .002**	.676 ± .004	.715 ± .008	.737 ± .003
Trace-handcrafted	.662 ± .003	**.777 ± .015**	**.741 ± .003**	**.759 ± .002**
Qwen3-4B	.562 ± .056	.607 ± .023	.636 ± .024	.653 ± .032
Gemma 3	.557 ± .063	.642 ± .041	.628 ± .037	.647 ± .040
MiniLM	.597 ± .031	.617 ± .031	.608 ± .043	.628 ± .011
SCPV	.559 ± .036	.586 ± .024	.622 ± .018	.676 ± .021
STV	.580 ± .013	.571 ± .022	.610 ± .009	.653 ± .019
Deep Sets	.591 ± .019	.620 ± .026	.595 ± .007	.668 ± .008
Set Transformer	.647 ± .010	.628 ± .009	.616 ± .036	.643 ± .015
GIN	.640 ± .006	.624 ± .023	.571 ± .028	.662 ± .026
GCN	.524 ± .059	.557 ± .051	.570 ± .040	.583 ± .045
GATv2	.549 ± .031	.573 ± .026	.540 ± .024	.591 ± .039
Graphormer	.556 ± .026	.566 ± .050	.520 ± .044	.497 ± .078
SAN	.516 ± .040	.479 ± .073	.503 ± .078	.495 ± .057
GT	.620 ± .012	.655 ± .021	.683 ± .020	.702 ± .014
DAGNN	.577 ± .048	.589 ± .013	.585 ± .030	.634 ± .018
PACE	.522 ± .051	.528 ± .062	.586 ± .067	.598 ± .055
DAGformer	.566 ± .052	.608 ± .037	.666 ± .057	.621 ± .057

Trace-only results. Tables 18 and 19 remove the time-series branch while retaining the same trace encoder, Tail-exceedance aggregation, and logistic head. This isolates the predictive signal available from traces alone.

Table 18: Trace-only FD AUPRC results. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best.

Model	FD-30s	FD-1m	FD-5m	FD-10m
Trace-zero	.159 ± .000	.159 ± .000	.159 ± .000	.159 ± .000
Trace-handcrafted	**.280 ± .011**	**.263 ± .004**	**.303 ± .006**	**.328 ± .020**
Qwen3-4B	.163 ± .010	.192 ± .011	.174 ± .025	.175 ± .027
Gemma 3	.216 ± .051	.199 ± .027	.155 ± .019	.166 ± .026
MiniLM	.187 ± .030	.175 ± .017	.155 ± .021	.182 ± .042
SCPV	.137 ± .007	.148 ± .009	.176 ± .022	.204 ± .012
STV	.151 ± .022	.146 ± .008	.156 ± .012	.190 ± .025
Deep Sets	.175 ± .006	.189 ± .010	.166 ± .013	.191 ± .025
Set Transformer	.166 ± .021	.179 ± .025	.176 ± .039	.166 ± .019
GIN	.176 ± .009	.189 ± .006	.155 ± .010	.185 ± .018
GCN	.162 ± .014	.173 ± .009	.179 ± .035	.179 ± .017
GATv2	.159 ± .017	.172 ± .023	.170 ± .013	.169 ± .023
Graphormer	.198 ± .026	.203 ± .044	.211 ± .020	.223 ± .063
SAN	.176 ± .045	.165 ± .029	.164 ± .037	.158 ± .037
GT	.187 ± .040	.195 ± .035	.235 ± .035	.222 ± .018
DAGNN	.190 ± .021	.200 ± .020	.214 ± .058	.213 ± .029
PACE	.181 ± .031	.197 ± .030	.203 ± .018	.214 ± .021
DAGformer	.228 ± .060	.182 ± .020	.187 ± .032	.168 ± .047

Table 19: Trace-only FD AUROC results. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best. Colors correspond to input formats (cf. Table 3).

Model	FD-30s	FD-1m	FD-5m	FD-10m
Trace-zero	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Trace-handcrafted	**.591 ± .009**	**.642 ± .003**	**.666 ± .010**	**.672 ± .014**
Qwen3-4B	.502 ± .019	.526 ± .028	.487 ± .031	.502 ± .075
Gemma 3	.521 ± .059	.505 ± .036	.457 ± .039	.480 ± .051
MiniLM	.513 ± .061	.485 ± .045	.468 ± .053	.487 ± .039
SCPV	.400 ± .025	.446 ± .032	.489 ± .042	.547 ± .041
STV	.457 ± .055	.437 ± .027	.450 ± .036	.503 ± .019
Deep Sets	.502 ± .008	.518 ± .030	.449 ± .025	.506 ± .034
Set Transformer	.480 ± .042	.495 ± .050	.474 ± .073	.464 ± .039
GIN	.512 ± .015	.492 ± .014	.435 ± .020	.486 ± .028
GCN	.478 ± .035	.472 ± .048	.479 ± .049	.475 ± .040
GATv2	.453 ± .040	.454 ± .025	.462 ± .016	.431 ± .031
Graphormer	.554 ± .043	.553 ± .074	.562 ± .039	.550 ± .102
SAN	.474 ± .062	.462 ± .083	.469 ± .087	.427 ± .055
GT	.508 ± .030	.538 ± .072	.575 ± .041	.524 ± .021
DAGNN	.513 ± .036	.547 ± .031	.517 ± .056	.536 ± .026
PACE	.502 ± .053	.526 ± .060	.556 ± .043	.545 ± .044
DAGformer	.580 ± .068	.514 ± .040	.504 ± .078	.465 ± .102

Threshold-dependent metrics. For each run, the decision threshold is selected on the validation split to maximize F1 and then applied unchanged to the test split. Tables 20 and 21 average each test metric across the four observation windows for each seed before aggregating across seeds.

Table 20: Threshold-dependent Trace + Time Series FD results. For each seed, each metric is averaged across the four post-deployment windows. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best. Colors correspond to input formats (cf. Table 3).

Model	Precision	Recall	F1	Accuracy
Task-specific references				
Majority	.159 ± .000	**1.000 ± .000**	.274 ± .000	.159 ± .000
Version-error delta	.437 ± .000	.201 ± .000	.206 ± .000	.808 ± .000
Time-series-only	.422 ± .025	.199 ± .004	.244 ± .002	.831 ± .003
Trace + Time Series				
Trace-zero	.421 ± .016	.214 ± .037	.252 ± .027	**.831 ± .004**
Trace-handcrafted	.460 ± .033	.338 ± .021	**.373 ± .013**	.828 ± .007
Qwen3-4B	.328 ± .035	.263 ± .080	.247 ± .042	.758 ± .037
Gemma 3	.394 ± .094	.194 ± .081	.209 ± .040	.789 ± .036
MiniLM	.348 ± .044	.201 ± .049	.233 ± .027	.796 ± .027
SCPV	.335 ± .011	.192 ± .026	.221 ± .018	.781 ± .014
STV	.339 ± .081	.190 ± .017	.221 ± .015	.785 ± .016
Deep Sets	.424 ± .016	.193 ± .023	.240 ± .018	.806 ± .008
Set Transformer	**.465 ± .056**	.183 ± .043	.236 ± .035	.818 ± .018
GIN	.403 ± .043	.172 ± .041	.227 ± .041	.817 ± .005
GCN	.243 ± .044	.303 ± .033	.232 ± .034	.688 ± .034
GATv2	.278 ± .040	.296 ± .035	.237 ± .034	.713 ± .020
Graphormer	.178 ± .026	.596 ± .279	.245 ± .041	.468 ± .180
SAN	.191 ± .021	.607 ± .240	.241 ± .031	.456 ± .164
GT	.405 ± .048	.260 ± .045	.297 ± .028	.812 ± .012
DAGNN	.260 ± .031	.325 ± .015	.276 ± .025	.724 ± .033
PACE	.198 ± .053	.768 ± .116	.262 ± .008	.357 ± .110
DAGformer	.402 ± .068	.271 ± .147	.236 ± .042	.745 ± .094

Table 21: Trace-only FD threshold-dependent results averaged across the four observation windows. For each seed, the threshold is selected on validation to maximize F1 and then applied to test. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best. Colors correspond to input formats (cf. Table 3).

Model	Precision	Recall	F1	Accuracy
Trace-zero	.159 ± .000	**1.000 ± .000**	.274 ± .000	.159 ± .000
Trace-handcrafted	**.318 ± .010**	.372 ± .084	**.313 ± .033**	**.761 ± .026**
Qwen3-4B	.158 ± .012	.325 ± .116	.197 ± .046	.629 ± .075
Gemma 3	.158 ± .012	.469 ± .129	.213 ± .019	.513 ± .103
MiniLM	.150 ± .014	.422 ± .093	.209 ± .019	.543 ± .067
SCPV	.163 ± .017	.619 ± .168	.246 ± .025	.421 ± .098
STV	.149 ± .009	.413 ± .124	.203 ± .026	.539 ± .077
Deep Sets	.163 ± .003	.464 ± .128	.230 ± .014	.519 ± .090
Set Transformer	.163 ± .014	.467 ± .094	.229 ± .016	.522 ± .089
GIN	.167 ± .016	.410 ± .029	.212 ± .010	.552 ± .032
GCN	.152 ± .012	.412 ± .085	.214 ± .015	.531 ± .066
GATv2	.142 ± .011	.464 ± .106	.210 ± .016	.467 ± .065
Graphormer	.190 ± .023	.660 ± .149	.270 ± .034	.464 ± .138
SAN	.157 ± .015	.688 ± .235	.244 ± .017	.350 ± .189
GT	.172 ± .017	.540 ± .083	.234 ± .030	.489 ± .045
DAGNN	.166 ± .010	.386 ± .102	.220 ± .015	.586 ± .077
PACE	.170 ± .019	.656 ± .194	.258 ± .027	.422 ± .164
DAGformer	.156 ± .014	.588 ± .115	.216 ± .010	.438 ± .093

D.9 Time-Series-Only Variates and Predictors

We vary the input variates and predictor used by Time-series-only. The model reported in the main results uses an identity time-series encoder and a linear logistic head. We compare it with LightGBM under the same train/test split. Each predictor receives either all five released variates (hits, errors, and P50, P95, and P99 latency), only hits and errors, or the derived error-rate series.

Table 22: **Time-series-only variates and predictors.** Mean AUPRC across the four FD observation windows ± sd over five seeds. Within each predictor, **bold** = best and underline = second-best.

Input time-series variates	LightGBM	Time-series-only
All variates	.247 ± .011	.295 ± .002
Hits + errors	**.250 ± .010**	.342 ± .003
Error rate	.249 ± .005	**.362 ± .001**

The error-rate configuration obtains the highest mean AUPRC for Time-series-only, which outperforms LightGBM for every tested input. This supports the configuration used in the main experiments.

D.10 Frontier LLM Evaluation

We evaluate Claude Opus 5 and GPT-5.6 Sol on FD in two modes. In-context inference supplies the observations directly in the prompt and requests a faulty-deployment score. Agentic inference supplies paths to the data, which the model can inspect using tools before returning its score. Each mode is evaluated with time series and traces, time series alone, and traces alone. On time series alone, the LLMs achieve 0.364–0.383 AUPRC, compared with 0.362 for the trained Time-series-only baseline. Adding traces does not improve either LLM's score in either mode. With traces alone, all four configurations underperform Trace-handcrafted (Table 18). The highest combined-input LLM score is 0.361, compared with 0.398 for Trace-handcrafted with Tail-exceedance (Figure 5). Agentic inference uses fewer reported tokens but takes longer for both models, averaging 54–73 seconds per instance compared with 12–27 seconds for in-context inference. By comparison, Trace-handcrafted requires only 7 ms per instance, illustrating the potential of compact and efficient trace representations for low-latency deployment monitoring.

Table 23: **Frontier LLMs and Trace-handcrafted on FD.** AUPRC is averaged across the four observation windows. Tokens and latency are per-instance averages with time series and traces. Token counts include input tokens for in-context inference and total tokens for agentic inference. Within each AUPRC column, **bold** = best and underline = second-best.

		AUPRC			Average per instance	
Model	Mode	Time series + traces	Time series only	Traces only	Tokens	Latency
Claude Opus 5	In-context	.328	.379	.161	629.2k	27 s
GPT-5.6 Sol	In-context	.361	.366	.182	528.0k	12 s
Claude Opus 5	Agentic	.303	**.383**	.199	99.7k	54 s
GPT-5.6 Sol	Agentic	.296	.364	.203	112.4k	73 s
Trace-handcrafted	Representation	**.398**	.362	**.294**	–	7 ms

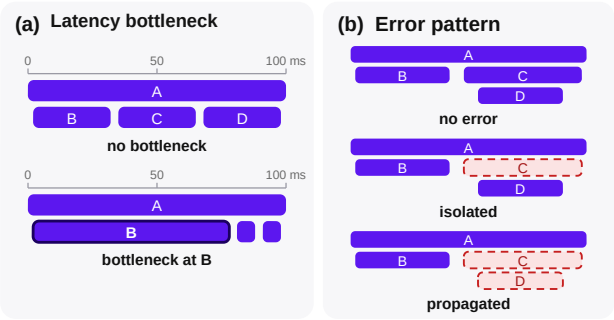


Figure 14: **The two intrinsic tasks, one example trace per label.**

E Intrinsic Task Details

E.1 Dataset Statistics

LB contains 15,755 labeled traces, comprising 11,650 *no bottleneck* and 4,105 *has bottleneck* examples. EP contains 39,000 labeled traces, comprising 31,252 *no error*, 3,874 *isolated*, and 3,874 *propagated* examples. Each source is split independently by root service, with no root service shared between train and test. Table 24 gives the source-level graph and split statistics.

Table 24: Intrinsic-task dataset statistics by source. Train and test counts refer to labeled traces.

Task	Source	#Traces	#Spans	#Edges	Train	Test
LB	Observability Company	9,735	144,090	134,355	7,788	1,947
LB	Uber	6,020	1,528,009	1,521,989	4,816	1,204
EP	Observability Company	20,000	549,433	529,433	16,000	4,000
EP	Uber	19,000	2,825,653	2,806,653	15,200	3,800

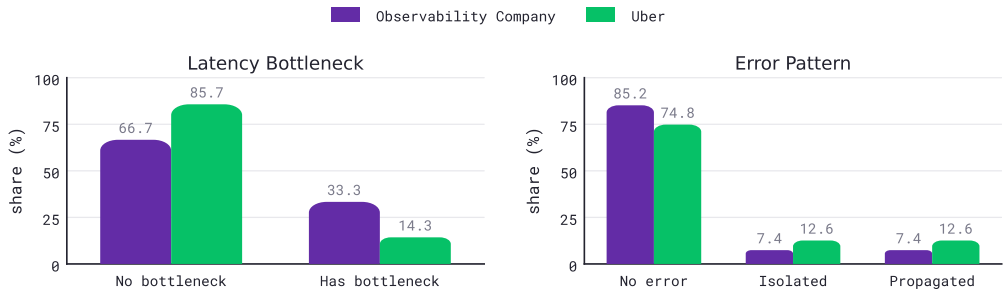


Figure 15: Intrinsic-task label distributions by source. Percentages are normalized independently within each task and source.

E.2 Latency Bottleneck

We derive the latency-bottleneck label from the critical path of each trace. Following CRISP, the critical path identifies the sequence of dependent computations that determines end-to-end latency (Zhang et al., 2022b). We use the recursive single-trace construction in CRISP's Listing 1. At each span, its children are sorted by decreasing end time. The last-finishing child is selected first. The remaining children are then selected in that order only when they finish before the previously selected child starts. The same procedure is applied recursively to every selected child. Our implementation uses the exact happens-before relation, $\mathrm{end}(c) \le \mathrm{start}(c')$.

Let $\mathcal{P}(x)$ denote the resulting critical-path spans of trace x, let r be its root span, let $d(s)$ be the duration of span s, and let $\mathrm{ch}(s)$ be its direct children. CRISP associates each critical-path span with an inclusive time, its wall-clock duration, and an exclusive time, the critical-path time not attributed to its selected callees. In our implementation, the exclusive time of $s \in \mathcal{P}(x)$ is

$$e_x(s) = d(s) - \sum_{c \in \mathrm{ch}(s) \cap \mathcal{P}(x)} d(c). \tag{9}$$

These exclusive times partition the root duration, so $\sum_{s \in \mathcal{P}(x)} e_x(s) = d(r)$. The root itself is not a bottleneck candidate. We measure the largest non-root contribution as

$$\kappa(x) = \frac{\max_{s \in \mathcal{P}(x) \setminus \{r\}} e_x(s)}{d(r)}, \tag{10}$$

and assign

$$y_{\mathrm{lat}}(x) = \begin{cases} \text{HAS BOTTLENECK}, & \kappa(x) > 0.5, \\ \text{NO BOTTLENECK}, & \text{otherwise.} \end{cases} \tag{11}$$

Because the exclusive times sum to $d(r)$, the strict threshold at 0.5 is equivalent to requiring one non-root span to contribute more exclusive time than all remaining critical-path spans combined.

E.3 Error Pattern

Let $L(x)$ be the longest consecutive run of error-status spans along any parent chain in trace x. The label is *no error* when $L(x) = 0$, *isolated* when $L(x) = 1$, and *propagated* when $L(x) \ge 2$. Thus propagation requires at least one error-status span whose direct parent also has error status. Errors separated by a non-error span do not form one propagated chain.

E.4 Extended Results

Task-specific baselines. Majority assigns every trace the empirical class-prior vector of the training source. EP additionally includes Error count, a single-feature logistic classifier based on the number of error-status spans. LB includes Span count and Max duration, single-feature logistic classifiers based on the number of spans and $\log(1 + d_{\max})$, where $d_{\max}$ is the maximum span duration. Each scalar feature is standardized on the training data, and the logistic classifiers use class-balanced training.

EP is a three-class task. Table 25 reports Macro AUPRC and Macro AUROC for every source pair. Table 26 additionally reports Macro Precision, Macro Recall, Macro F1, and accuracy, averaging each seed across the four source pairs before computing summary statistics.

Table 25: EP results across trace sources. Arrows indicate train–test source pairs, where Obs denotes the observability-company corpus. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best.

	Macro AUPRC				Macro AUROC			
Model	Obs→Obs	Obs→Uber	Uber→Obs	Uber→Uber	Obs→Obs	Obs→Uber	Uber→Obs	Uber→Uber
Baselines								
Majority	.333 ± .000	.333 ± .000	.333 ± .000	.333 ± .000	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Error count	.667 ± .000	.667 ± .000	.667 ± .000	.667 ± .000	.973 ± .000	.952 ± .000	**.973 ± .000**	.952 ± .000
Qwen3-4B	.754 ± .020	.552 ± .019	.562 ± .009	.706 ± .005	.975 ± .002	.859 ± .016	.858 ± .011	.929 ± .005
Gemma 3	.577 ± .002	.374 ± .001	.507 ± .001	.453 ± .001	.899 ± .001	.568 ± .001	.774 ± .002	.682 ± .001
MiniLM	.455 ± .002	.375 ± .006	.381 ± .000	.390 ± .000	.774 ± .005	.585 ± .010	.600 ± .000	.602 ± .000
SCPV	.333 ± .000	.333 ± .000	.333 ± .000	.333 ± .000	.499 ± .002	.500 ± .000	.500 ± .000	.500 ± .000
STV	.333 ± .000	.333 ± .000	.333 ± .000	.333 ± .000	.495 ± .011	.500 ± .000	.500 ± .000	.500 ± .000
Deep Sets	.525 ± .076	.364 ± .036	.551 ± .065	.871 ± .072	.732 ± .070	.526 ± .092	.787 ± .091	**.965 ± .036**
Set Transformer	.712 ± .082	.641 ± .099	.338 ± .016	.459 ± .098	.931 ± .091	.903 ± .082	.487 ± .069	.684 ± .062
GIN	.563 ± .139	.354 ± .018	.614 ± .101	.634 ± .151	.746 ± .137	.522 ± .060	.725 ± .096	.827 ± .099
GCN	.553 ± .048	.413 ± .047	.579 ± .087	.828 ± .034	.824 ± .061	.624 ± .068	.749 ± .079	.926 ± .021
GATv2	.716 ± .082	.663 ± .135	.623 ± .124	.853 ± .031	.914 ± .032	.867 ± .072	.806 ± .106	.928 ± .019
Graphormer	.656 ± .056	.669 ± .096	.619 ± .146	.761 ± .173	.900 ± .075	.872 ± .115	.841 ± .181	.875 ± .155
SAN	.383 ± .016	.403 ± .006	.398 ± .035	.446 ± .098	.652 ± .059	.587 ± .009	.640 ± .055	.628 ± .086
GT	.462 ± .053	.376 ± .066	.352 ± .021	.479 ± .048	.711 ± .072	.510 ± .066	.532 ± .065	.728 ± .039
DAGNN	**.957 ± .032**	**.999 ± .002**	**.876 ± .056**	**.941 ± .048**	**.992 ± .007**	**1.000 ± .000**	.955 ± .023	.964 ± .031
PACE	.785 ± .085	.764 ± .087	.697 ± .146	.908 ± .046	.976 ± .012	.954 ± .017	.917 ± .059	.964 ± .024
DAGformer	.720 ± .064	.535 ± .096	.718 ± .105	.733 ± .096	.921 ± .015	.784 ± .120	.929 ± .053	.925 ± .042

Table 26: EP threshold-dependent results. For each seed, metrics are averaged across the four train–test source pairs. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best.

Model	Macro Precision	Macro Recall	Macro F1	Accuracy
Baselines				
Majority	.267 ± .000	.333 ± .000	.296 ± .000	.800 ± .000
Error count	.583 ± .000	.667 ± .000	.572 ± .000	.900 ± .000
Qwen3-4B	.647 ± .045	.653 ± .012	.595 ± .029	.842 ± .019
Gemma 3	.465 ± .007	.469 ± .008	.356 ± .011	.549 ± .014
MiniLM	.278 ± .026	.370 ± .015	.284 ± .018	.569 ± .007
SCPV	.127 ± .098	.333 ± .000	.155 ± .099	.380 ± .293
STV	.173 ± .052	.333 ± .000	.202 ± .053	.520 ± .157
Deep Sets	.589 ± .031	.611 ± .019	.540 ± .041	.765 ± .120
Set Transformer	.417 ± .064	.447 ± .042	.383 ± .066	.714 ± .164
GIN	.531 ± .073	.556 ± .066	.436 ± .093	.653 ± .193
GCN	.569 ± .036	.589 ± .031	.494 ± .017	.716 ± .077
GATv2	.695 ± .076	.618 ± .089	.602 ± .098	.858 ± .039
Graphormer	.669 ± .088	.636 ± .114	.600 ± .100	.836 ± .100
SAN	.293 ± .126	.385 ± .022	.229 ± .049	.388 ± .091
GT	.321 ± .063	.397 ± .045	.288 ± .032	.566 ± .032
DAGNN	**.884 ± .040**	**.856 ± .028**	**.833 ± .046**	**.935 ± .054**
PACE	.675 ± .038	.710 ± .032	.649 ± .036	.876 ± .047
DAGformer	.584 ± .108	.578 ± .048	.502 ± .085	.726 ± .119

LB is binary, with *has bottleneck* as the positive class. Tables 27 and 28 report AUPRC and AUROC by source pair and the threshold-dependent metrics, respectively.

Table 27: LB results across trace sources. Arrows indicate train–test source pairs, where Obs denotes the observability-company corpus. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best. Colors correspond to input formats (cf. Table 3).

	AUPRC				AUROC			
Model	Obs→Obs	Obs→Uber	Uber→Obs	Uber→Uber	Obs→Obs	Obs→Uber	Uber→Obs	Uber→Uber
Baselines								
Majority	.333 ± .000	.143 ± .000	.333 ± .000	.143 ± .000	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Span count	.333 ± .000	.143 ± .000	.333 ± .000	.143 ± .000	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Max duration	.336 ± .000	.143 ± .000	.336 ± .000	.143 ± .000	.502 ± .000	.487 ± .000	.502 ± .000	.487 ± .000
Qwen3-4B	.357 ± .005	.157 ± .010	.324 ± .002	.160 ± .006	.536 ± .008	.503 ± .023	.479 ± .005	.528 ± .010
Gemma 3	.335 ± .002	.139 ± .002	.332 ± .004	.143 ± .006	.509 ± .001	.492 ± .010	.497 ± .007	.503 ± .019
MiniLM	.339 ± .009	.147 ± .014	.335 ± .006	.149 ± .008	.500 ± .011	.506 ± .019	.492 ± .006	.524 ± .016
SCPV	.333 ± .000	.143 ± .000	.333 ± .000	.143 ± .000	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
STV	.333 ± .000	.143 ± .000	.333 ± .000	.143 ± .000	.500 ± .000	.500 ± .000	.500 ± .000	.500 ± .000
Deep Sets	.341 ± .006	.145 ± .008	.329 ± .009	.162 ± .010	.508 ± .009	.498 ± .005	.486 ± .017	.510 ± .011
Set Transformer	.335 ± .010	.162 ± .027	.328 ± .008	.148 ± .007	.507 ± .012	.507 ± .011	.482 ± .013	.499 ± .017
GIN	.350 ± .007	.154 ± .010	.325 ± .011	.154 ± .016	.521 ± .010	.499 ± .004	.489 ± .008	.514 ± .020
GCN	.347 ± .021	.141 ± .004	.316 ± .010	.154 ± .011	.519 ± .015	.485 ± .016	.470 ± .005	.512 ± .020
GATv2	.342 ± .012	.141 ± .008	.323 ± .011	.152 ± .011	.515 ± .013	.493 ± .019	.481 ± .017	.514 ± .026
Graphormer	.351 ± .008	**.177 ± .024**	**.339 ± .011**	.154 ± .012	.521 ± .013	**.549 ± .032**	**.503 ± .017**	.508 ± .018
SAN	.337 ± .004	.142 ± .001	.337 ± .005	.148 ± .004	.502 ± .003	.499 ± .002	.500 ± .005	.501 ± .001
GT	.337 ± .013	.143 ± .004	.322 ± .008	.141 ± .003	.501 ± .015	.501 ± .015	.480 ± .010	.490 ± .009
DAGNN	**.395 ± .039**	.155 ± .022	.332 ± .002	**.246 ± .017**	**.555 ± .029**	.487 ± .019	.483 ± .005	**.644 ± .029**
PACE	.342 ± .012	.147 ± .006	.333 ± .005	.183 ± .026	.513 ± .013	.496 ± .010	.500 ± .010	.548 ± .050
DAGformer	.344 ± .010	.166 ± .021	.336 ± .002	.233 ± .041	.515 ± .012	.527 ± .040	.502 ± .003	.617 ± .073

Table 28: LB threshold-dependent results. For each seed, metrics are averaged across the four train–test source pairs. Values are mean ± sample standard deviation over five seeds. **bold** = best, underline = second-best. Colors correspond to input formats (cf. Table 3).

Model	Precision	Recall	F1	Accuracy
Baselines				
Majority	.000 ± .000	.000 ± .000	.000 ± .000	**.762 ± .000**
Span count	.000 ± .000	.000 ± .000	.000 ± .000	**.762 ± .000**
Max duration	.241 ± .000	.557 ± .000	.246 ± .000	.409 ± .000
Qwen3-4B	.247 ± .003	.334 ± .142	.197 ± .055	.602 ± .088
Gemma 3	.072 ± .066	.220 ± .230	.103 ± .095	.648 ± .120
MiniLM	.024 ± .053	.100 ± .224	.037 ± .084	.709 ± .117
SCPV	.095 ± .053	.400 ± .224	.150 ± .084	.552 ± .117
STV	.167 ± .065	**.700 ± .274**	.263 ± .103	.395 ± .143
Deep Sets	.080 ± .058	.097 ± .095	.046 ± .035	.696 ± .066
Set Transformer	.071 ± .065	.300 ± .274	.113 ± .103	.605 ± .143
GIN	.142 ± .094	.166 ± .154	.123 ± .091	.679 ± .086
GCN	.179 ± .074	.344 ± .314	.188 ± .148	.582 ± .166
GATv2	.131 ± .086	.300 ± .188	.148 ± .084	.603 ± .098
Graphormer	.071 ± .106	.239 ± .336	.093 ± .131	.630 ± .189
SAN	.071 ± .065	.249 ± .250	.092 ± .094	.623 ± .132
GT	.048 ± .106	.200 ± .447	.075 ± .168	.657 ± .234
DAGNN	.256 ± .034	.600 ± .276	**.310 ± .062**	.474 ± .149
PACE	.171 ± .070	.456 ± .287	.228 ± .094	.534 ± .138
DAGformer	**.278 ± .037**	.458 ± .191	.265 ± .066	.530 ± .111

F Encoder and Implementation Details

F.1 Trace Inputs

Figure 16 summarizes the four inputs derived from each trace. Set-based and graph encoders represent `service`, `operation_name`, and `resource_name` with learned 32-dimensional embeddings. They concatenate one-hot encodings of `status`, `span_kind`, `type`, and `language` with span duration after a $\log(1 + x)$ transform and start time min–max normalized within each trace. Categorical vocabularies are fitted on the corresponding training split. General graph encoders use symmetrized parent–child edges, while DAG-aware encoders retain their direction.

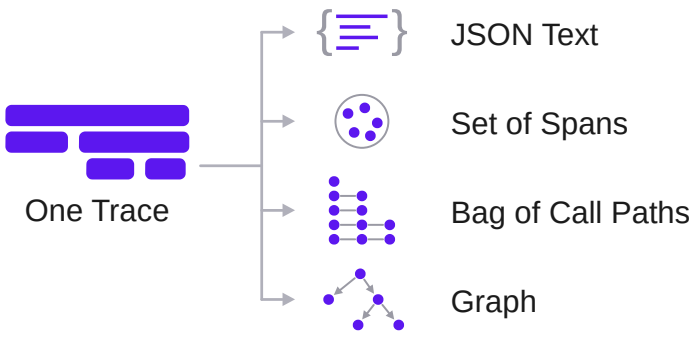


Figure 16: The input views derived from each trace.

F.2 Encoder Configurations

Table 29: Encoder configurations. Parameters exclude task-specific projection and prediction heads. Colors correspond to input formats (cf. Table 3).

Encoder	Parameters	Output dim.	Pretrained	Configuration
MiniLM (2020; 2019)	22.7M	384	Yes	{chunk: 256, overlap: 64, token pool: mean, chunk pool: weighted mean}
Gemma 3 (2025)	300M	768	Yes	{chunk: 2,048, overlap: 256, token pool: mean, chunk pool: weighted mean}
Qwen3-Embedding-4B (2025)	4B	2,560	Yes	{chunk: 32,768, overlap: 256, token pool: last, chunk pool: weighted mean}
Deep Sets (2017)	1.48M	128	No	{ϕ layers: 4, ρ layers: 4, hidden dim.: 384, pool: sum}
Set Transformer (2019)	1.45M	128	No	{ISAB blocks: 5, heads: 8, inducing points: 32, PMA seeds: 1}
GCN (2017)	0.60M	256	No	{layers: 4, hidden dim.: 128, pool: mean}
GATv2 (2022)	1.07M	256	No	{layers: 4, hidden dim.: 128, heads: 4, pool: mean}
GIN (2019)	0.80M	256	No	{layers: 4, hidden dim.: 128, pool: JK-sum}
GT (2021)	1.84M	128	No	{layers: 10, heads: 8, Laplacian eigenvectors: 8, norm: batch, pool: mean}
Graphormer (2021)	43.8M	768	No	{layers: 12, heads: 16, max. spatial distance: 20, pool: virtual node}
SAN (2021)	2.17M	128	No	{layers: 10, heads: 8, Laplacian eigenvectors: 10, norm: batch, pool: sum}
DAGNN (2021)	0.97M	128	No	{layers: 2, propagation: bidirectional, aggregation: attention, pool: max endpoint}
PACE (2022)	1.07M	128	No	{layers: 3, heads: 8, pool: virtual sink}
DAGformer (2023)	1.71M	128	No	{layers: 4, heads: 8, extractor: 3-hop, encoding: depth, pool: mean}

Gemma 3 denotes the `google/embeddinggemma-300m` checkpoint. For the language encoders, only task-specific downstream components are trained. In TR, Qwen3-Embedding-4B encodes the query for every non-language trace encoder. Language configurations share their respective encoder across the query and trace towers.

F.3 Task-Specific Optimization

All neural experiments ran on NVIDIA A100 GPUs. Table 30 gives the task-level training configurations.

Table 30: Task-specific optimization configurations.

Task	Objective	Head	Optimizer	Learning rate	Max epochs
TR	Margin ranking, $m = 1$	Linear projections	AdamW	$[10^{-4}, 7\times10^{-4}]$	50
FD	Class-weighted BCE	Linear	AdamW	$[10^{-5}, 3\times10^{-2}]$	100
LB	Class-weighted CE	Linear	AdamW	$[10^{-4}, 10^{-3}]$	50
EP	Class-weighted CE	Linear	AdamW	$[10^{-4}, 10^{-3}]$	50