TRACECODEC: A COMPILER-BACKED NEURAL CODEC FOR NETWORK TRAFFIC TRACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Critical networking workflows require high-fidelity packet captures (PCAPs) for testing, security analysis, and protocol validation, not just statistical flow-level summaries. Recent packet generators have demonstrated protocol-constrained PCAP synthesis, but many operate on raw or near-raw packet representations. These representations can entangle learned behavioral choices with deterministic protocol rules and make packet correctness depend on post-hoc repair. We identify the decode interface as a key bottleneck and present TRACECODEC, a compiler-backed neural codec for stateful multi-flow traces. TRACECODEC lifts each parsed packet into a timed packet action with explicit flow slots and transport cues, learns a continuous per-packet latent, and uses a state-aware deterministic compiler to decode actions to PCAPs while managing endpoint assignment, TCP sequence and acknowledgment state, declared compatibility constraints, and packet rendering. On CICIDS2017 Monday, TRACECODEC matches packet count, protocol composition, and flow population to within 0.03%; raw-field baselines produce substantially larger flow-count and TCP-event errors under the same non-repair policy. In a matched flow-matching continuation setup, TRACECODEC yields complete decoded suffixes in 100% of CICIDS2017 cases and 98.44% of MAWI cases, versus at most 10.94% for raw-field latent controls. Structural diagnostics further show that TRACECODEC preserves TCP transitions and multi-flow interleaving. Together, these results support TRACECODEC as a generator-facing latent interface with a deterministic path back to usable packet traces.[1]

1 INTRODUCTION

Synthetic network traffic matters because real network traces can be very hard to share due to privacy or governance restrictions, or costly to collect (Jiang et al., 2024; Chu et al., 2026; Zhou et al., 2026). Synthetic generation can also support network evaluation and improve coverage of diverse, rare, or adversarial cases (Cüppers et al., 2024; Jin & Apostolaki, 2025; Gupta et al., 2025). Yet useful synthetic traffic is not merely a statistically plausible table of flow summaries. Many workflows consume packet-level artifacts: traces are inspected and processed with tools such as Wireshark, DPDK, and Scapy (Sikos, 2020; DPDK Project, 2026; Biondi & community, 2026), transformed into downstream features (Holland et al., 2021; Liu et al., 2023a), replayed in timing-sensitive environments (Turner & Klassen, 2024), and used to test systems under protocol constraints. In these settings, PCAP is the object that downstream systems actually consume.

Recent work has established that packet-level generation is a realistic target. Jiang et al. (2024) introduce NetDiffusion for protocol-constrained packet-trace generation, and Chu et al. (2026) introduce NetSSM for longer, multi-flow, state-aware generation. In parallel, pretrained traffic representation models such as ET-BERT, TrafficFormer, netFound, and TrafficLLM show that useful traffic semantics can be learned for downstream analysis and cross-task generalization (Lin et al., 2022; Zhou et al., 2025; Beltiukov et al., 2026; Cui et al., 2025). These advances leave open a different question: *once packet traces are treated as generative targets, what representation should a packet-trace generator operate on?*

The core problem is the decode interface. Raw packet representations mix behavioral choices with fields whose values depend on flow history and packet-rendering rules. Asking one learned decoder

[1]The code is available in an anonymous repository.

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.

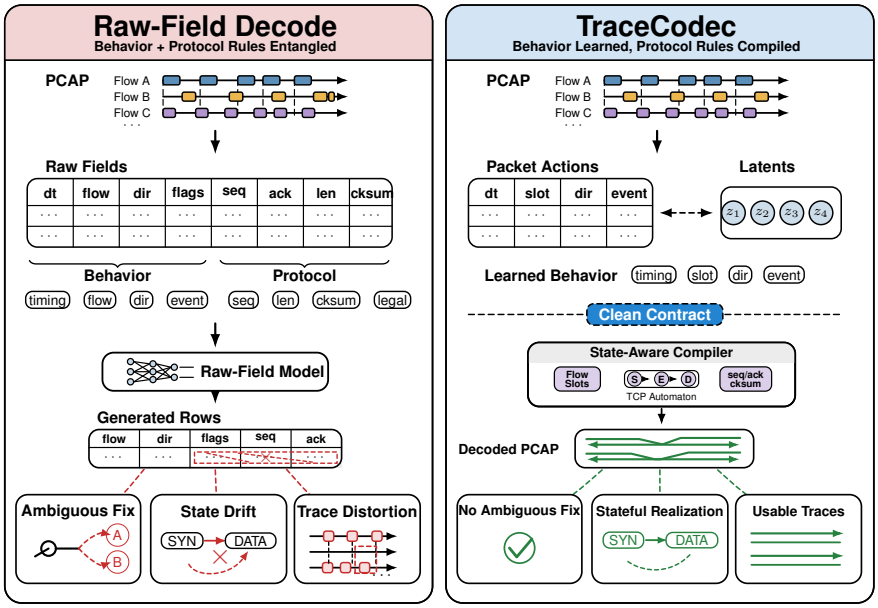


Figure 1: **Comparison of Decode Interfaces.** Raw-field decoding can entangle behavioral choices with protocol consequences, leaving consistency dependent on post-hoc repair. TraceCodec learns PacketActions and realizes them under explicit flow and transport state.

to predict both can produce internally inconsistent packets and make final realization depend on post-hoc repair (Chu et al., 2026; Jiang et al., 2024). A generator-facing codec therefore needs a clear contract between learned packet behavior and state-dependent realization.

More challenges arise in three properties that make packet traces usable. **(1) Timing.** Inter-arrival times determine bursts, throughput, and latency distributions. Existing systems handle timing differently: NetSSM assigns timestamps post hoc from a ground-truth distribution, whereas netFound embeds inter-arrival and burst metadata for representation learning (Chu et al., 2026; Beltiukov et al., 2026). A generator-facing codec instead needs an explicit timing contract. **(2) Network protocol statefulness.** TCP packets are constrained by a state machine: flags, sequence numbers, acknowledgments, and payload lengths must agree with the current flow state. Existing generators either learn these dependencies within the generator or apply post-generation correction, rather than exposing an explicit state-aware decode contract (Jiang et al., 2024; Chu et al., 2026). **(3) Multiple concurrent flows.** A capture may contain many interleaved flows, so a generator must preserve flow identity across the packet sequence. NetSSM's multi-flow evaluation, for example, reports three to five typically active flows in its Netflix workload (Chu et al., 2026).

Figure 1 previews our approach. We introduce TRACECODEC, a compiler-backed neural codec that uses *PacketActions* rather than raw packet fields as its decode interface. Each PacketAction records the behavioral choices represented at one packet step. *For timing*, we pair every action with an inter-packet time delta. *For transport-state progression*, actions retain direction, TCP control behavior, payload length, and selected progression cues, while a state-aware deterministic compiler maintains endpoint assignment and sequence and acknowledgment state. *For multi-flow identity*, actions refer to reusable flow slots, and the compiler maintains the state associated with each active slot. The neural codec itself remains packet-aligned, mapping each timed PacketAction to one continuous latent vector.

This separation gives downstream generative models a continuous per-packet latent space and a deterministic path back to parseable, timed, stateful PCAPs. Across decoded-PCAP evaluations, TRACECODEC stays close to the reference on packet and timing statistics, while raw-field baselines incur much larger flow-count and TCP-event errors. Structural diagnostics further show that it preserves TCP transitions and multi-flow structure.

Contributions. This paper makes the following contributions:

- We identify the codec-interface problem for packet-trace generation and argue that generators should disentangle behavioral choices from deterministic protocol constraints.
- We introduce TRACECODEC, a compiler-backed neural codec built around timed PacketActions and reusable flow slots. A packet-aligned VAE maps each action and time delta to a continuous latent, while a state-aware deterministic compiler realizes decoded actions under explicit flow and transport state.
- We evaluate TRACECODEC as a packet-trace codec on three public datasets. After rendering reconstructed outputs to PCAP, it achieves higher packet-, timing-, transport-, and flow-level fidelity than the evaluated raw-field baselines. Under a matched downstream generator, its latent sequences also produce complete PCAP continuations substantially more often than raw-field latent controls.

2 BACKGROUND AND RELATED WORK

Traffic representations. Traffic representations span byte-level and segment-level encoders for classification and protocol understanding (Lotfollahi et al., 2020; Wang et al., 2017; Liu et al., 2019; Xiao et al., 2022) and structured packet, flow, and graph views (Holland et al., 2021; Meng et al., 2022; van Ede et al., 2020; Huoh et al., 2023). Recent systems pretrain representations over payload, datagram, flow, and cross-layer inputs (He et al., 2021; Lin et al., 2022; Zhao et al., 2023; Peng et al., 2024; Wang et al., 2024). PACC studies protocol-aware compression (Guo et al., 2026), while TrafficFormer, netFound, and TrafficLLM pursue broader transfer and general-purpose traffic representations (Zhou et al., 2025; Beltiukov et al., 2026; Cui et al., 2025). These models target analysis and transfer rather than a generator-facing codec interface whose decoder must materialize timed PCAPs while maintaining flow identity and protocol state.

Packet generators. Packet generators instead synthesize network artifacts. NetShare generates IP-header traces (Yin et al., 2022); NetDiffusion synthesizes packet-like representations with protocol-constrained realization (Jiang et al., 2024); and NetSSM models longer, multi-flow raw-packet sequences while leaving timestamps outside the learned sequence (Chu et al., 2026). These systems establish native packet generation, but their primary contract is an end-to-end generator rather than a reusable paired codec. TRACECODEC addresses this complementary interface problem.

Codec interfaces. Learned codecs make high-dimensional generation tractable in images and audio (van den Oord et al., 2017; Rombach et al., 2022; Zeghidour et al., 2021). Packet traces impose an additional requirement: decoding must materialize timed packets under evolving protocol and multi-flow state. TRACECODEC brings the codec abstraction to this setting through PacketActions and deterministic state realization. Table 1 summarizes the resulting interface distinction.

3 TRACECODEC DESIGN

3.1 OVERVIEW

TRACECODEC separates packet behavior from its state-dependent realization. First, state-aware lifting maps each parsed packet and the current lifting state to a PacketAction and an inter-packet time delta, then advances the lifting state. Second, a packet-aligned VAE maps each timed action to one continuous latent vector. The VAE has no recurrent state or attention across packet steps. Third, a state-aware deterministic compiler executes reconstructed actions against explicit multi-flow decode state and emits concrete packets.

Figure 2 shows this division of responsibility. Lifting converts concrete packet fields and capture-side flow state into timed PacketActions. Flow slots preserve interaction identity in the resulting action sequence. During decoding, the compiler binds those slots to concrete flows and advances their protocol state. Downstream generators operate only on the latent sequence between these two stateful transformations.

Table 1: Positioning of TRACECODEC relative to traffic representations and packet generators.

Work	Learned object	PCAP decode	Explicit timing	State / multi-flow	Realization contract
TRAFFIC REPRESENTATIONS					
ET-BERT (Lin et al., 2022)	datagram tokens	not required	not required	not a decode state	none
TrafficFormer (Zhou et al., 2025)	traffic tokens	not required	order, not explicit timing	not a decode state	none
netFound (Beltiukov et al., 2026)	traffic representation	not required	not required	not a decode state	none
TrafficLLM (Cui et al., 2025)	field/value tokens	not required	not required	not a decode state	none
PACKET GENERATORS					
NetDiffusion (Jiang et al., 2024)	packet-like image	yes, native	not directly generated	single-flow traces	protocol-constrained post-processing
NetSSM (Chu et al., 2026)	raw packet sequence	yes, native	post hoc; reference-derived	implicit in generator state	direct generation
PACKET-TRACE CODEC					
TraceCodec (ours)	per-packet latent over timed PacketActions	**yes**	**yes**	**flow slots; state in compiler**	**deterministic compiler under explicit state**

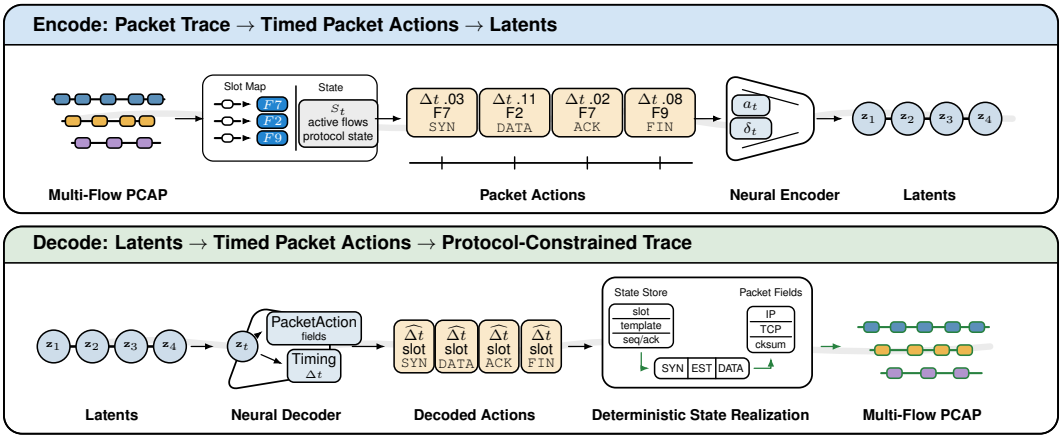


Figure 2: Encoding and decoding phases of TRACECODEC

3.2 PACKET ACTIONS

A *PacketAction* is a structured record of the behavioral choices represented at one packet step. It records the flow slot, protocol, direction, control event, payload behavior, and selected transport cues. It omits fields reconstructed from decode state, including concrete endpoints and absolute TCP sequence numbers. A PacketAction is therefore a codec IR, not a compressed packet header.

Lifting packets. After ordinary PCAP parsing, the lifter processes each eligible parsed packet against the current lifting state. It maps the packet's concrete flow to a reusable slot, derives the selected relative transport cues, emits a discrete PacketAction $\mathbf{a}_t$ and an inter-packet time delta δ_t, and then advances the lifting state. The action and time delta form one packet-aligned codec record. Table 2 lists the discrete action fields.

The `flow_token` field names a reusable slot in a bounded active-flow vocabulary, not an endpoint identifier. It preserves which active interaction the packet belongs to while leaving concrete endpoint assignment to the compiler.

The `flow_evt` field controls the slot lifecycle. NEW initializes a slot for a flow, CONTINUE refers to the active flow in that slot, and END releases the slot after its final packet. A generation index distinguishes successive flows that reuse the same slot.

Table 2: PacketAction schema. Rows group fields only for presentation; each packet step uses one fixed-width action record.

Group	Fields and decode role
Flow, family, and event identity	`flow_token`, `flow_evt`, `proto`, `ip_family`, `dir`. Identify the bounded active-flow slot, protocol path, IP family, and flow lifecycle role.
TCP control semantics	`tcp_ctrl`. Represent TCP control events such as SYN, SYN-ACK, FIN, and RST without requiring raw flag prediction.
Transport fields	`tcp_opt_profile`, `tcp_win_d0..d3`, `tcp_ack_adv_d0..d7`, `tcp_ack_unseen`, `tcp_seq_delta_sign`, `tcp_seq_delta_d0..d7`. Encode option profile, receive-window value, bounded acknowledgment advancement, unseen-ACK state, and bounded sequence progression.
Packet fields	`ttl_res`, `icmp_type`, `icmp_code`, `l4_payload_len_d0..d3`. Encode IP-family-specific values, ICMP control fields, and layer-4 payload length.

Behavioral variables. The IR retains variables whose changes affect the decoded trace. Packet order is preserved by the action sequence, timing by δ_t, and flow identity by reusable slots. A decoded PacketAction does not determine every field of the realized packet. Concrete endpoints and absolute TCP sequence and acknowledgment numbers depend on decode state; derived lengths and checksums are recomputed during rendering. Application payload semantics remain outside the codec.

State-aware compilation. A PacketAction is therefore a residual rather than a complete packet: the packet-aligned VAE reconstructs each timed action independently, while the compiler combines it with the current decode state to emit a packet and advance that state. Section 3.3 formalizes this transition.

3.3 Packet Compiler

The compiler executes one decoded action at a time. The global decode state $\mathbf{s}_t$ contains the active flow slots and their protocol state. Given this state, the compiler emits one packet and advances the state:

$$(\hat{\mathbf{p}}_t, \mathbf{s}_{t+1}) = \mathcal{C}(\mathbf{s}_t, \hat{\mathbf{a}}_t, \hat{\delta}_t).$$

Starting from an empty state, repeated application emits the reconstructed trace within the codec scope. The compiler realizes state-dependent consequences through FlowTable lookup, transport-state updates, and packet rendering; it is part of the decode contract, not a repair stage applied after generation.

Flow templates. The compiler first resolves the slot encoded by `flow_token`. A deterministic FlowTable binds the slot and its generation index to a synthetic endpoint template. The decoded direction selects the endpoint orientation. Reusable slots give the interface a fixed active-flow capacity without making the VAE reconstruct raw five-tuples.

Transport state. For TCP traffic, the compiler maintains per-flow, per-direction transport state. It assigns initial sequence numbers, tracks sequence and acknowledgment progress, and updates the state after each decoded control or data event. The PacketAction supplies the event and its bounded progression cues; the compiler performs the corresponding arithmetic.

UDP and ICMP require less state, but their realization remains explicit. The compiler instantiates endpoints and applies the decoded direction, payload length, and ICMP control fields. In every protocol path, the VAE predicts bounded behavior and the compiler constructs the packet under explicit state.

Constrained rendering. Before rendering TCP packets, the compiler applies a fixed set of local compatibility constraints. It clears payload length on control events and acknowledgment advancement on a pure SYN. These rules resolve only the declared conflicts inside an action. They do not correct an incorrect protocol, flow event, slot, direction, or TCP control event.

After flow and transport interpretation, the compiler renders a concrete packet using link, network, and transport headers. Payload bytes are deterministic synthetic material by default; their lengths are controlled by the decoded action, but the bytes themselves do not reconstruct application-level content. TRACECODEC therefore targets parseable, transport-consistent packet traces rather than byte-identical application conversations.

3.4 LATENT CODEC

With the PacketAction IR and deterministic compiler fixed, the learned component is a packet-aligned VAE (Kingma & Welling, 2014). Each forward pass processes one PacketAction and its time delta. The VAE has no recurrent hidden state and no attention across packet steps. A trace becomes an ordered latent sequence with one vector per packet step.

Formulation. Let $\mathbf{a}_t$ denote a PacketAction and δ_t its inter-packet time delta. The encoder defines

$$q_\phi(\mathbf{z}_t \mid \mathbf{a}_t, \delta_t),$$

and the decoder defines

$$p_\theta(\mathbf{a}_t, \delta_t \mid \mathbf{z}_t).$$

The decoder produces $\hat{\mathbf{a}}_t$ and $\hat{\delta}_t$; protocol-state transitions remain the compiler's responsibility.

The encoder embeds the fields of one timed PacketAction as tokens, applies a compact Transformer within that record, and maps the result to Gaussian latent parameters. Field-specific classification heads reconstruct the fixed-width PacketAction, and a separate regression head reconstructs timing. Neither the encoder nor the decoder carries state across packet steps. We use a conventional backbone so that the evaluation focuses on the representation and decode interface.

Latent interface. The continuous latent layer is the interface exposed to downstream generators. It keeps timing with packet behavior and gives a sequence model one vector per packet step. Other latent families can use the same PacketAction IR because the compiler contract does not depend on the VAE architecture.

Training objective. The model is trained with a variational objective

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \lambda_{\text{time}}\mathcal{L}_{\text{time}} + \beta\mathcal{L}_{\text{KL}},$$

where $\mathcal{L}_{\text{recon}}$ aggregates field-wise reconstruction losses over the PacketAction record, $\mathcal{L}_{\text{time}}$ measures timing reconstruction error, and $\mathcal{L}_{\text{KL}}$ regularizes the latent space. Reconstruction is computed in packet-action space rather than packet space; decoded packets are obtained only after passing reconstructed packet actions through the deterministic compiler.

The KL term regularizes the generator-facing latent space; the ablation study separates this role from local reconstruction fidelity.

4 EVALUATION

4.1 EXPERIMENTAL SETUP

Datasets and baselines. We evaluate TRACECODEC on CICIDS2017 Monday (Sharafaldin et al., 2018) and MAWI `202004071400` (Cho et al., 2000), which together cover enterprise and backbone traffic. To broaden the evaluation to a distinct VPN/non-VPN application-traffic setting, we additionally evaluate on VNAT (Jorgensen et al., 2024). Raw-field baselines use four published tabular generative models adapted to the same encode–decode task: TVAE (Xu et al., 2019), TabSyn-VAE (Zhang et al., 2024), GOGGLE (Liu et al., 2023b), and TTVAE (Wang & Nguyen, 2025).

Evaluation protocol. We render every raw-field baseline output under a shared non-repair policy: packet-local recomputation is allowed, but endpoint inference, sequence-level repair, TCP-state repair, and use of TRACECODEC's compiler are not. We also include an *Oracle* control, which lowers ground-truth PacketActions and timing through the deterministic compiler without a learned bottleneck. All neural codecs are trained from scratch; none uses an external pretrained backbone.

Table 3: Decoded-PCAP fidelity on held-out test sets. Lower is better; bold marks the best learned method, and "–" denotes no valid output. Oracle is a nonlearned control.

Dataset	Method	TCP event dist. (pp) ↓	Packet count err. (%) ↓	Protocol TV (%) ↓	IAT TV (%) ↓	Flow count err. (%) ↓	Flow duration TV (%) ↓
CICIDS2017							
Monday	TVAE	72.73	37.83	5.16	14.83	25.19	45.36
	TabSyn-VAE	66.74	2.97	0.62	1.08	46.32	34.76
	GOGGLE	–	–	–	–	–	–
	TTVAE	53.19	37.65	5.34	14.40	104.98	51.03
	TRACECODEC	**14.51**	**0.00**	**0.00**	**0.84**	**0.03**	**9.23**
	Oracle	2.17	0.00	0.00	0.04	0.00	0.01
MAWI							
202004071400	TVAE	52.67	14.21	8.06	29.33	91.31	29.87
	TabSyn-VAE	53.63	28.05	6.71	5.92	76.07	10.73
	GOGGLE	–	–	–	–	–	–
	TTVAE	50.37	61.96	18.27	26.45	81.29	17.54
	TRACECODEC	**2.31**	**0.00**	**0.00**	**0.38**	**0.07**	**0.25**
	Oracle	1.55	0.00	0.00	0.01	0.00	0.00
VNAT	TVAE	113.68	51.14	3.32	47.64	61.02	57.09
	TabSyn-VAE	74.96	4.45	0.51	7.13	30.07	48.38
	GOGGLE	–	–	–	–	–	–
	TTVAE	122.86	31.38	2.03	59.14	384.63	55.51
	TRACECODEC	**0.14**	**0.00**	**0.00**	**1.08**	**8.69**	**6.84**
	Oracle	0.00	0.00	0.00	0.02	0.00	0.00

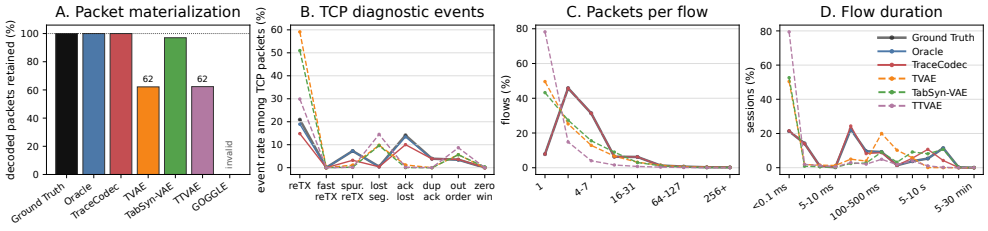


Figure 3: Protocol state preservation analysis on CICIDS2017 Monday.

All decoded-PCAP metrics are computed after packet rendering. Count columns report relative errors, TV columns compare distributions, and TCP-event distance compares diagnostic-event rate profiles rather than invalid-packet rates. Dataset construction and splitting, baseline adaptation, model capacity, runtime, training budgets, and metric definitions are given in Appendices A, C, D, and F.

4.2 PCAP RECONSTRUCTION FIDELITY

Table 3 evaluates fully rendered PCAPs. The Oracle has zero packet-count, protocol, and flow-count error on all three datasets, showing that the symbolic path retains this structure before learning. After neural reconstruction, TRACECODEC has zero count and protocol error on all three datasets; its flow-count errors are 0.03%, 0.07%, and 8.69% on CICIDS2017, MAWI, and VNAT. The largest remaining errors are TCP events and flow duration on CICIDS2017 and flow count on VNAT. Raw-field baselines instead lose packets, fragment flows, or distort transport behavior. Figure 3 exposes these failure modes through packet materialization, TCP diagnostics, and flow distributions.

4.3 TCP TRANSITIONS AND MULTI-FLOW INTERLEAVING

TCP transition consistency. Marginal event rates do not capture event ordering within a session. Figure 4 therefore measures event-to-event transitions within bidirectional sessions. The Oracle and learned TRACECODEC have weighted row-wise total-variation distances of 3.66×10^{-5} and 2.76×10^{-5} from the reference, versus 0.229–0.353 for the raw-field baselines. Thus, TRACECODEC closely matches the reference distribution of TCP-event transitions beyond their marginal rates.

Multi-flow interleaving. Figure 5 tests whether decoded packets preserve concurrent sessions without spurious fragmentation. The reference, Oracle, and learned TRACECODEC have similar

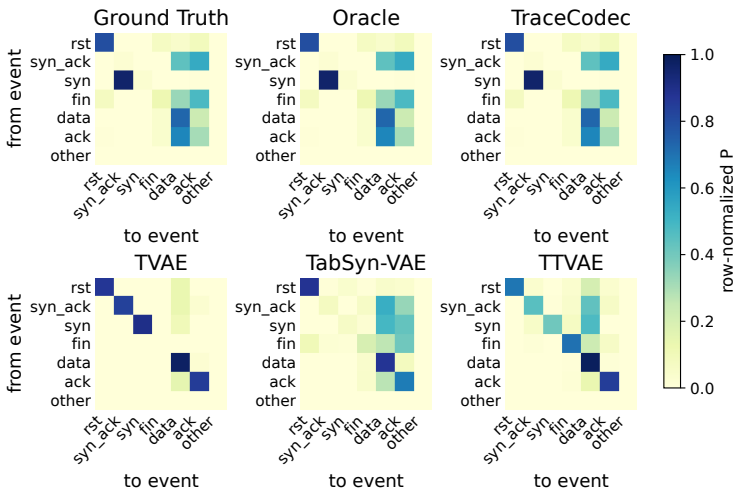


Figure 4: TCP transition and state consistency on CICIDS2017 Monday. Each heatmap is a row-normalized transition matrix over coarse TCP event labels within bidirectional sessions.

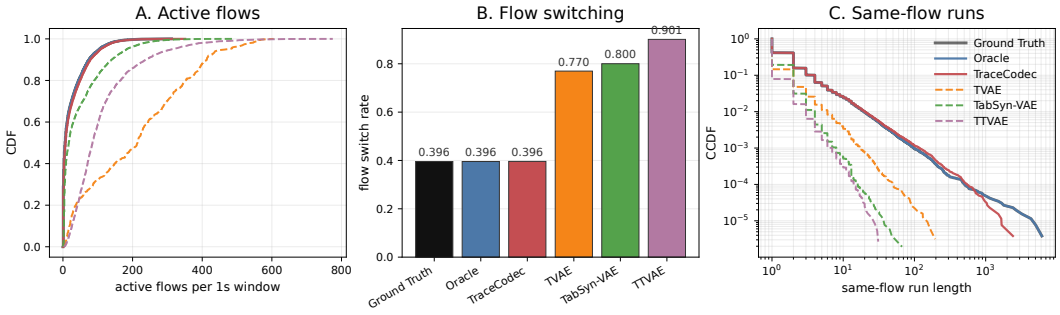


Figure 5: Active-flow and multi-flow interleaving on CICIDS2017 Monday.

flow-switch rates (≈ 0.396) and same-flow run lengths (mean ≈ 2.53). Raw-field baselines switch flows more often, with rates of 0.770–0.901, and inflate or lose sessions. TRACECODEC therefore retains the reference interleaving pattern without fragmenting flows.

4.4 LATENT INTERFACE FOR TRAFFIC CONTINUATION

This experiment evaluates the usability of the latent decode interface, not a complete end-to-end traffic-generation system. We use the same conditional flow-matching architecture (Lipman et al., 2023) and protocol across representations. The models contain 135.15–135.22M parameters, a difference below 0.05%. For each matched window, the generator observes a 1024-token prefix and produces a 1024-token suffix, which we decode and compare with the held-out suffix PCAP. We include the standalone TVAE and TabSyn-VAE latents; TTVAE requires input-derived encoder memory, and GOGGLE produces no valid rows under the shared non-repair contract.

We count a suffix as usable when its rendered PCAP contains exactly the requested number of packets; valid-packet coverage and distributional fidelity are reported separately. Coverage-adjusted total variation (CA-TV) compares each decoded distribution with its reference while charging missing or invalid packet mass. Ordinary TV can instead make a method appear accurate after it drops difficult packets.

Table 4 shows that TRACECODEC produces usable suffixes in all 128 cases with 100% median valid-packet coverage, compared with 0% usable suffixes for TVAE and 7.03% for TabSyn-VAE. It

Table 4: Latent-interface results on CICIDS2017 Monday over 128 matched windows. Entries are case-level medians except usable suffixes; CA denotes coverage adjustment.

Representation	Usable suffixes (%) ↑	Valid pkt coverage (%) ↑	CA IAT TV (%) ↓	CA pkt-size TV (%) ↓	CA flow-pkts TV (%) ↓	CA sess-dur TV (%) ↓
TRACECODEC	**100.00**	**100.00**	**13.00**	**18.85**	**39.35**	**40.44**
TVAE	0.00	63.18	41.07	49.76	64.80	57.75
TabSyn-VAE	7.03	98.29	16.09	25.59	42.99	42.59

Table 5: Codec and realization ablations on CICIDS2017 Monday. The upper block compares codec variants against its full-model control. The lower block compares two realizers on the same frozen codec reconstruction and belongs to a separate paired run. Boxes mark targeted effects; bold marks the better realizer; "–" denotes a metric not reported for that block. KL rate is in nats per packet.

Variant / realizer	Action field acc. (%) ↑	TCP event dist. (pp) ↓	IAT TV (%) ↓	Flow count err. (%) ↓	Flow duration TV (%) ↓	Valid pkt coverage (%) ↑	KL rate (nats/pkt)
CODEC COMPONENTS							
Full TRACECODEC	99.73	13.79	0.59	0.40	8.41	–	167.91
W/O KL	99.58	14.57	0.49	0.41	8.64	–	5222.77
W/O FINE TIMING	99.77	12.89	13.37	0.40	5.90	–	152.41
W/O TCP CUES	99.95	138.85	0.51	0.40	8.10	–	94.22
DETERMINISTIC VS. LEARNED REALIZATION							
Deterministic compiler	–	**12.66**	**0.49**	**0.04**	**9.13**	**100.00**	–
Causal LSTM realizer	–	43.52	85.89	631.22	77.61	89.75	–

also has the lowest coverage-adjusted timing and structural distances. Continuation remains imperfect, but sampled TRACECODEC latents materialize as complete PCAPs far more consistently.

4.5 CODEC AND REALIZATION ABLATIONS

Codec components. The upper block of Table 5 separates timing, TCP progression and window cues, and latent regularization within a matched ablation run. Its full-model row is the control for these variants and is not the separate reconstruction artifact reported in Table 3. Coarsening timing raises IAT TV from 0.59% to 13.37%. Removing TCP progression and window cues leaves action accuracy at 99.95% but raises TCP-event distance to 138.85 percentage points. Removing KL changes reconstruction little but raises the posterior KL rate from 167.91 to 5222.77 nats per packet. Thus, action accuracy alone does not establish timing fidelity, transport behavior, or latent regularity.

Realization mechanism. Because PacketActions omit state-dependent packet fields, TRACECODEC uses a deterministic compiler for realization. We compare it with a 4.27M-parameter causal LSTM that receives the same frozen decoded actions, input time deltas, packet order, and PCAP boundaries, but predicts complete raw packet fields, including output timing. The LSTM raises TCP-event distance from 12.66 to 43.52 percentage points, IAT TV from 0.49% to 85.89%, and flow-count error from 0.04% to 631.22%; valid-packet coverage falls from 100% to 89.75%. Under this shared action interface, the evaluated learned realizer does not match deterministic compilation. Appendix E.1 details its targets, paired results, and limitations.

5 CONCLUSION

This paper frames packet-trace representation learning as a codec-interface problem. PacketActions record packet behavior and timing without asking the neural codec to reproduce state-dependent packet fields. A packet-aligned VAE represents this interface, while a state-aware deterministic compiler realizes each decoded action under protocol state. Our experiments show that this interface reduces the reconstruction failures observed for the evaluated raw-field baselines. TRACECODEC does not reconstruct application-layer payload semantics, but it provides a direct path from generator-facing latents to timed packet traces realized under explicit flow and transport state.

AI USE STATEMENT

Generative AI assistance was used during manuscript preparation for language editing, LaTeX organization, and consistency checks. The authors reviewed and verified the resulting text, equations, citations, and experimental claims and remain responsible for the submitted manuscript.

ETHICS STATEMENT

This work uses publicly available network-trace datasets and does not involve human subjects, user studies, or the collection of new personally identifying information. The codec suppresses raw endpoint identifiers and payload bytes and uses synthetic endpoint and payload realization. Privacy-relevant field handling and its limitations are discussed in Appendix B.

REPRODUCIBILITY STATEMENT

The code and run entry points are provided in the anonymous repository linked in the paper. Public dataset sources, temporal splits, preprocessing, training budgets, hardware, and evaluation commands are documented in Appendices A and F; the main experimental artifacts are generated from the shared scripts and recorded run manifests.

REFERENCES

Sylee Beltiukov, Satyandra Guthula, Haarika Manda, Jaber Daneshamooz, Wenbo Guo, Walter Willinger, Arpit Gupta, and Inder Monga. netFound: Principled design for network foundation models, 2026. URL https://arxiv.org/abs/2310.17025. arXiv preprint arXiv:2310.17025, version 5.

Philippe Biondi and Scapy community. Scapy: Manipulate packets. https://scapy.net/, 2026. Accessed: 2026-05-06.

Kenjiro Cho, Kenji Mitsuya, and Akira Kato. Traffic data repository at the WIDE project. In *Proceedings of the 2000 USENIX Annual Technical Conference, FREENIX Track*, pp. 263–270, 2000. URL https://mawi.wide.ad.jp/mawi/.

Andrew Chu, Xi Jiang, Shinan Liu, Arjun Nitin Bhagoji, Francesco Bronzino, Paul J. Schmitt, and Nick Feamster. NetSSM: Multi-flow and state-aware network trace generation using state-space models. *Proceedings of the ACM on Networking*, 4(CoNEXT1), 2026. doi: 10.1145/3786289. URL https://doi.org/10.1145/3786289.

Tianyu Cui, Xinjie Lin, Sijia Li, Miao Chen, Qilei Yin, Qi Li, and Ke Xu. TrafficLLM: Enhancing large language models for network traffic analysis with generic traffic representation, 2025. URL https://arxiv.org/abs/2504.04222. arXiv preprint arXiv:2504.04222.

Joscha Cüppers, Adrien Schoen, Gregory Blanc, and Pierre-Francois Gimenez. FlowChronicle: Synthetic network flow generation through pattern set mining. *Proceedings of the ACM on Networking*, 2(CoNEXT4):1–20, 2024. doi: 10.1145/3696407. URL https://doi.org/10.1145/3696407.

DPDK Project. DPDK – The open source data plane development kit accelerating network performance. https://www.dpdk.org/, 2026. Accessed: 2026-05-06.

Zhaochen Guo, Tianyufei Zhou, Honghao Wang, Ronghua Li, and Shinan Liu. PACC: Protocol-aware cross-layer compression for compact network traffic representation, 2026. URL https://arxiv.org/abs/2602.08331. arXiv preprint arXiv:2602.08331.

Ragini Gupta, Shinan Liu, Ruixiao Zhang, Xinyue Hu, Xiaoyang Wang, Hadjer Benkraouda, Pranav Kommaraju, Phuong Cao, Nick Feamster, and Klara Nahrstedt. Generative active adaptation for drifting and imbalanced network intrusion detection. *arXiv preprint arXiv:2503.03022*, 2025. URL https://arxiv.org/abs/2503.03022.

Hongye He, Zhiguo Yang, and Xiangning Chen. Payload encoding representation from transformer for encrypted traffic classification. *ZTE Communications*, 19 (4):90–97, 2021. URL https://www.zte.com.cn/global/about/magazine/zte-communications/2021/en202104/researchpaper/en202104010.html.

Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3366–3383, 2021. doi: 10.1145/3460120.3484758.

Ting-Li Huoh, Yan Luo, Peilong Li, and Tong Zhang. Flow-based encrypted network traffic classification with graph neural networks. *IEEE Transactions on Network and Service Management*, 20 (2):1224–1237, 2023. doi: 10.1109/TNSM.2022.3227500.

Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul J. Schmitt, Francesco Bronzino, and Nick Feamster. NetDiffusion: Network data augmentation through protocol-constrained traffic generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 8(1), 2024. doi: 10.1145/3639037. URL https://doi.org/10.1145/3639037.

Minhao Jin and Maria Apostolaki. Robustifying ML-powered network classifiers with PANTS. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 7291–7310. USENIX Association, 2025. URL https://www.usenix.org/conference/usenixsecurity25/presentation/jin-minhao.

Steven Jorgensen, John Holodnak, Jensen Dempsey, Karla de Souza, Ananditha Raghunath, Vernon Rivet, Noah DeMoes, Andrés Alejos, and Allan Wollaber. Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification. *IEEE Transactions on Artificial Intelligence*, 5(1):420–433, 2024. doi: 10.1109/TAI.2023.3244168.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014. URL https://arxiv.org/abs/1312.6114.

Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification, 2022. URL https://arxiv.org/abs/2202.06335. arXiv preprint arXiv:2202.06335.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. FS-Net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019 – IEEE Conference on Computer Communications*, pp. 1171–1179, 2019. doi: 10.1109/INFOCOM.2019.8737507.

Shinan Liu, Tarun Mangla, Ted Shaowang, Jinjin Zhao, John Paparrizos, Sanjay Krishnan, and Nick Feamster. Amir: Active multimodal interaction recognition from video and network traffic in connected environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 7(1):1–26, 2023a.

Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. GOGGLE: Generative modelling for tabular data by learning relational structure. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=fPVRcJqspu.

Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020. doi: 10.1007/s00500-019-04030-2.

Xuying Meng, Yequan Wang, Runxin Ma, Haitong Luo, Xiang Li, and Yujun Zhang. Packet representation learning for traffic classification. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3546–3554, 2022. doi: 10.1145/3534678.3539085.

Lingfeng Peng, Xiaohui Xie, Sijiang Huang, Ziyi Wang, and Yong Cui. PTU: Pre-trained model for network traffic understanding. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pp. 1–12, 2024. doi: 10.1109/ICNP61940.2024.10858503.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pp. 108–116, 2018. doi: 10.5220/0006639801080116. URL https://www.unb.ca/cic/datasets/ids-2017.html.

Leslie F Sikos. Packet analysis for network forensics: A comprehensive survey. *Forensic Science International: Digital Investigation*, 32:200892, 2020.

Aaron Turner and Fred Klassen. Tcpreplay: Pcap editing and replaying utilities. https://github.com/appneta/tcpreplay, 2024. Accessed: 2026-05-06.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, 2017.

Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David R. Choffnes, Maarten van Steen, and Andreas Peter. FlowPrint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020. doi: 10.14722/ndss.2020.24412.

Alex X. Wang and Binh P. Nguyen. TTVAE: Transformer-based generative modeling for tabular data generation. *Artificial Intelligence*, 340:104292, 2025. doi: 10.1016/j.artint.2025.104292. URL https://www.sciencedirect.com/science/article/pii/S0004370225000116.

Tongze Wang, Xiaohui Xie, Wenduo Wang, Chuyi Wang, Youjian Zhao, and Yong Cui. NetMamba: Efficient network traffic classification via pre-training unidirectional mamba. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pp. 1–11, 2024. doi: 10.1109/ICNP61940.2024.10858569.

Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 43–48, 2017. doi: 10.1109/ISI.2017.8004872.

Xi Xiao, Wentao Xiao, Rui Li, Xiapu Luo, Haitao Zheng, and Shutao Xia. EBSNN: Extended byte segment neural network for network traffic classification. *IEEE Transactions on Dependable and Secure Computing*, 19(5):3521–3538, 2022. doi: 10.1109/TDSC.2021.3101311.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *Advances in Neural Information Processing Systems 32*, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/254ed7d2de3b23ab10936522dd547b78-Abstract.html.

Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical GAN-based synthetic IP header trace generation using NetShare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 458–472, 2022. doi: 10.1145/3544216.3544251.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2021.

Hengrui Zhang, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4Ay23yeuz0.

Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5420–5427, 2023. doi: 10.1609/AAAI.V37I4.25674.

Guangmeng Zhou, Xiongwen Guo, Zhuotao Liu, Tong Li, Qi Li, and Ke Xu. TrafficFormer: An efficient pre-trained model for traffic data. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 1844–1860, 2025. doi: 10.1109/SP61157.2025.00102.

Yajie Zhou, Fuheng Zhao, Eric Wang, Ayse K. Coskun, Divyakant Agrawal, Amr El Abbadi, and Zaoxing Liu. PrvTel: Lightweight models for private and accurate telemetry data retention. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/zhou-yajie.

A Experimental Details

A.1 Published baselines and source traces

We compare TraceCodec against four published generic tabular models, each adapted to the shared raw packet-table interface described in Appendix C.

TVAE (Xu et al., 2019) is the variational autoencoder baseline introduced alongside CTGAN for mixed-type tabular data. It learns a continuous row latent and reconstructs numerical and categorical columns directly.

TabSyn-VAE (Zhang et al., 2024) is the VAE backbone used by TabSyn before latent-space diffusion. It tokenizes mixed-type rows and learns a Transformer-based latent representation; in our experiments, we use this VAE stage as a standalone codec rather than the full TabSyn generator.

GOGGLE (Liu et al., 2023b) is a graph-based tabular generator that learns relational structure among columns and decodes rows through a learned graph model.

TTVAE (Wang & Nguyen, 2025) is a Transformer-based variational autoencoder for tabular data. It replaces the MLP-style tabular backbone with a Transformer encoder–decoder.

The three source datasets contain public packet captures from distinct traffic regimes:

- **CICIDS2017 Monday** (Sharafaldin et al., 2018) is the benign Monday WorkingHours trace from the CICIDS2017 intrusion-detection dataset released by the Canadian Institute for Cybersecurity. We use it as an enterprise-style TCP-heavy setting with clear session structure.
- **MAWI `202004071400`** (Cho et al., 2000) is a public 15-minute backbone trace from the MAWI Working Group Traffic Archive. We use the samplepoint-F capture collected on 2020-04-07 14:00 as the higher-concurrency backbone setting.
- **VNAT** (Jorgensen et al., 2024) contains VPN and non-VPN captures from multiple network applications. We use it to test the codec on application traffic outside the enterprise and backbone settings above.

A.2 Datasets and split policy

All evaluation settings are built from raw packet traces rather than pre-aggregated flow summaries. For CICIDS2017 Monday and MAWI, we run a session-preserving temporal sharder, sort the resulting shard PCAPs by start time, and assign contiguous train/validation/test ranges over the ordered shard list. This keeps held-out shards temporally disjoint while preserving whole sessions inside each sample. The MAWI sharder produces 13,102 ordered shards; we use the first 3,000 and split them into 2,400/300/300 train, validation, and test shards.

For VNAT, we use 107/9/12 disjoint source captures for training, validation, and testing. After applying the shared packet-level scope, the three partitions contain 6.53M, 0.81M, and 1.27M PacketActions, respectively. Table 6 summarizes all three experimental splits.

Table 6: Dataset statistics for the selected experimental splits after applying the shared packet-level scope. PCAPs are session-preserving shards for CICIDS2017 and MAWI and source captures for VNAT.

Dataset	PCAPs	Split PCAPs			Tokens			Protocol mix		
		Train	Val	Test	Train	Val	Test	TCP	UDP	ICMP
CICIDS2017 Monday	2,397	1,919	239	239	10.32M	0.68M	0.66M	92.0%	8.0%	0.02%
MAWI 202004071400	3,000	2,400	300	300	39.23M	1.70M	1.46M	48.4%	33.0%	18.5%
VNAT	128	107	9	12	6.53M	0.81M	1.27M	40.4%	59.6%	0.02%

A.3 Shared preprocessing and decode scope

All learned methods consume the same train, validation, and test partitions and are evaluated only after decoding their own outputs back to PCAP. The tokenizer, PacketAction schema, compiler, payload policy, and active-flow budget are fixed before training. Packets outside the shared packet-/transport-level decode scope are excluded consistently and recorded in the dataset manifests. Raw-field baselines use the same partitions but decode through the restricted renderer in Appendix C, so they do not inherit TraceCodec's state-aware compiler.

Ablation construction. The coarse-timing variant quantizes positive inter-packet deltas into eight bins that are uniform after the transform $\log(1 + \delta/\tau)$. Here, τ is the median positive training delta, and the upper endpoint is the 99.9th percentile. The transport-cue ablation removes relative TCP sequence and acknowledgment progression and the advertised-window fields, while retaining the TCP option profile. All other PacketAction fields and the compiler are unchanged.

A.4 Architecture and model capacity

All neural codecs are trained from scratch on each dataset, without an external pretrained backbone. Saved model paths refer only to locally trained checkpoints. TraceCodec uses the same architecture on all three datasets. Its encoder embeds fields into 128-dimensional tokens and applies a six-layer, eight-head Transformer independently to each packet record. Each Transformer layer has feed-forward width 1536 and dropout 0.15. The resulting record representation is projected through hidden width 768 to the mean and log variance of a 128-dimensional Gaussian latent. Field-specific classification heads reconstruct the fixed-width PacketAction, and a separate regression head reconstructs timing. The deterministic compiler contains no trainable parameters.

Table 7: Trainable parameter counts for the evaluated neural codecs. Counts are taken from the selected checkpoints. The compiler is deterministic and has no trainable parameters.

Method	CICIDS2017 Monday	MAWI	VNAT
TVAE	6,321,644	4,691,342	379,646
TabSyn-VAE	1,627,481	70,159,385	177,716
GOGGLE	14,791,699	631,578,835	1,743,814
TTVAE	8,435,780	6,805,478	2,493,782
TraceCodec	**11,494,401**	**11,494,401**	**11,494,401**
Compiler	0	0	0

Table 7 reports exact trainable parameter counts from the evaluated checkpoints. The raw-field baseline counts vary across datasets because their learned categorical vocabularies depend on the observed packet fields. Model size therefore does not track decoded-PCAP fidelity: for example, on MAWI, TabSyn-VAE and GOGGLE are respectively 6.1 and 55.0 times larger than TraceCodec, while TraceCodec is larger than several baselines on VNAT. In the latent-interface experiment, the downstream flow-matching models contain 135,217,281 parameters for TraceCodec, 135,215,744 for TVAE, and 135,154,264 for TabSyn-VAE. Their maximum parameter-count difference is below 0.05%.

Table 8: Runtime and resource use across datasets. Training times are approximate. Neural throughput is measured on one NVIDIA RTX PRO 6000 GPU; compile/render throughput uses one Intel Xeon Platinum 8480C CPU process.

Measurement	**CICIDS2017 Monday**	**MAWI**	**VNAT**
Training time for 50k steps (h)	13.0	12.2	11.4
Neural encode (k packets/s)	110.7	110.5	110.7
Neural decode (M packets/s)	2.84	2.82	2.84
Stateful compile/render (k packets/s)	1.48	1.40	1.75
Peak GPU allocation (GiB)	9.06	9.06	9.06

A.5 Runtime and resource use

We report each reconstruction stage separately because neural inference is batched, whereas packet realization follows capture order. Training uses one NVIDIA RTX PRO 6000 GPU with bfloat16 and includes validation and checkpointing. Neural throughput is measured over the first 120,000 PacketActions from each test split with batch size 30,000, after one 30,000-packet warm-up batch. The input tensors are already resident on the GPU; checkpoint loading, host-to-device transfer, and disk I/O are excluded.

Neural encoding sustains 110.5–110.7k packets/s, and neural decoding reaches 2.82–2.84M packets/s. The ordered compile/render stage processes 1.40–1.75k packets/s in one CPU process and is the main measured cost in the current reconstruction implementation. This measurement includes action materialization, flow and transport-state updates, packet construction, and byte serialization, but excludes writing bytes to PCAP files and subsequent TShark analysis. It uses a 5,000-packet warm-up followed by six 20,000-packet measurement chunks. A 50,000-step training run takes approximately 11.4–13.0 hours and reaches a peak GPU allocation of 9.06 GiB.

The compiler operates online in capture order. Its mutable state is bounded by the configured number of active flow slots rather than the total trace length.

B Privacy Considerations

This paper is not a formal privacy paper. It does not provide differential privacy, immunity to memorization, or jurisdiction-specific legal compliance. The privacy goal is specific: reduce direct disclosure of endpoint identifiers and application content while preserving the packet- and transport-level behavior needed by the codec.

B.1 Identifier suppression

TraceCodec keeps the variables needed for packet-/transport-level decode, such as timing, flow-slot identity, protocol, direction, control semantics, bounded acknowledgment advancement, receive-window structure, and payload-length behavior, but it does not learn raw endpoint identifiers or application payload bytes. In particular, the learned representation omits raw IP addresses, raw transport ports, MAC addresses, checksums, absolute TCP sequence and acknowledgment numbers, and payload content.

The practical implication is that the training representation is not a lightly disguised copy of the original packet headers. It is a structured behavioral representation whose purpose is to preserve packet- and transport-level semantics while suppressing directly identifying fields. Table 9 summarizes what is kept and what is omitted.

B.2 Synthetic realization

The decode path is synthetic rather than restorative. Flow templates are generated from flow-slot identity, protocol, IP family, generation index, and an optional salt; the resulting endpoints are therefore derived synthetic templates rather than original source and destination tuples. In the current implementation, IPv4 decode produces private-address-like endpoints, IPv6 decode produces unique-

Table 9: Privacy-relevant field treatment in TRACECODEC. The codec keeps behavior needed for packet-/transport-level fidelity while suppressing direct identifiers and payload content.

Trace Element	Treatment in TRACECODEC
Raw IP addresses, transport ports, MAC addresses	Not part of the learned packet-action representation; decode re-instantiates synthetic endpoints from flow-slot identity rather than restoring original identifiers.
Application payload bytes	Not learned. Decoded packets carry deterministic synthetic payload material by default, and truncated-capture output is also available when analysis-only artifacts are sufficient.
Absolute TCP sequence/ack numbers, checksums	Not learned. They are reconstructed deterministically from relative state and packet rendering rules during decode.
Fidelity-relevant behavior: timing, direction, protocol, control events, payload length, bounded transport detail	Retained, because these variables define the packet-/transport-level semantics the codec is intended to preserve.

local-style endpoints, transport ports are regenerated, MAC addresses are synthetic, and TCP initial sequence numbers are re-instantiated from deterministic hashed state rather than copied from the source trace.

Payload handling follows the same principle. Payload bytes are not recovered from the training trace. Instead, the renderer emits deterministic synthetic payload material of the required length, and an analysis-oriented truncated-capture mode can further suppress payload bytes in the written PCAP while preserving wire length. The implementation also supports randomized per-run flow salts so that the same latent code can be decoded into different synthetic endpoint assignments across runs, improving unlinkability between decoded traces and any specific original address assignment.

B.3 SCOPE

These choices reduce direct identifier exposure and prevent literal replay of the original application payload. They are therefore meaningful privacy-aware design choices, and they are one reason we treat application-semantic replay as out of scope for the paper. However, they do not amount to a proof of privacy. Sensitive information can still survive indirectly through timing structure, payload-length patterns, rare protocol behavior, or distinctive interaction dynamics. A determined adversary could also study model memorization or nearest-neighbor leakage at the level of whole traces rather than individual identifiers.

For that reason, the correct interpretation of TRACECODEC is not "privacy guaranteed," but "designed to suppress direct identifiers and payload content while preserving the semantics required for packet-/transport-level fidelity." Stronger privacy statements would require dedicated evaluation, such as membership-inference tests, nearest-neighbor disclosure audits, or formal mechanisms like differential privacy. Likewise, legal and institutional compliance remains a property of the full data-governance process, not of the model architecture alone.

C RAW-FIELD BASELINES

The main text compares TRACECODEC against published generic tabular models adapted to raw-packet roundtrip. This appendix specifies that adaptation in enough detail to make the fairness assumptions explicit.

C.1 RAW PACKET TABLE

Each packet is first converted into a mixed-type row that retains concrete header fields and timing rather than packet-action semantics. The design goal is not bit-identical reconstruction, but the strongest raw-field interface we can make compatible with protocol-constrained packet rendering

Table 10: Raw packet-table schema used by tabular baselines.

Group	Columns and role
Timing	`delta_t`. First-class inter-packet timing variable.
Network / packet identity	`ip_version`, `proto`, `src_ip`, `dst_ip`, `ttl`, `dscp`, `ecn`, `ip_flags`, `ip_frag_off`, `payload_len`. Keep concrete family, endpoint, header, and payload-length fields needed for local packet rendering.
TCP block (TCP rows only)	`tcp_sport`, `tcp_dport`, `tcp_seq`, `tcp_ack`, `tcp_flags`, `tcp_window`, `tcp_opt_profile`. Keep concrete transport values without introducing relative or state-derived semantics.
UDP block (UDP rows only)	`udp_sport`, `udp_dport`.
ICMP block (ICMP rows only)	`icmp_type`, `icmp_code`.

without silently introducing stateful repair. The table therefore keeps fidelity-relevant concrete packet fields and omits obviously derived values that can be recomputed locally during rendering.

For the auxiliary raw-table diagnostics, we separate three pre-render quantities. Invalid-row rate counts decoded rows that cannot be interpreted as valid raw packet rows under the shared schema, for example because an inactive-token leaks into an active field or because the decoded endpoints are not valid IP literals for the declared IP family. On the remaining decoded rows that satisfy the shared contract, we report categorical-field accuracy over active categorical columns and numerical-field error over active numerical columns in raw field units. These are local diagnostics rather than decoded-PCAP metrics. Their purpose is to show whether a raw-field baseline has learned a non-trivial packet-row reconstruction map before rendering is judged, without conflating pre-render contract violations with later packet-tool failures.

The resulting schema is summarized in Table 10. It is a direct per-packet projection of the protocol-neutral canonical packet contract used by the implementation rather than a bit-level dump or an engineered traffic-feature table. Source and destination IP addresses and transport ports are therefore retained as high-cardinality categorical columns, while `delta_t`, fragment offset, payload length, TCP sequence and acknowledgment numbers, and TCP window remain numerical columns.

Transport-specific columns are active only for rows whose `proto` value selects the matching block. Inactive categorical columns take a reserved "not applicable" token, while inactive numerical columns are set to zero and masked out of the local row-space diagnostics. The baseline therefore sees the same concrete packet fields that were present in the source packet and nothing more.

We exclude several tempting classes of columns. Locally derived quantities such as checksums, `ip_total_len`, `ip_payload_len`, `udp_len`, and redundant presence bits are omitted because they can be recomputed once the retained columns are fixed. Derived cross-packet variables such as flow IDs, canonicalized direction, retransmission labels, and relative TCP progression are omitted because they would add sequence or state information to the raw-field baseline. Application payload bytes are excluded to match the packet-/transport-level scope of this study.

C.2 Baseline methods

We use four published tabular generative models as the primary raw-field baselines: TVAE (Xu et al., 2019), TabSyn-VAE (Zhang et al., 2024), GOGGLE (Liu et al., 2023b), and TTVAE (Wang & Nguyen, 2025). Each has a public implementation and a direct enough reconstruction path to support packet-wise encode–decode without redefining the task. In this paper they are evaluated narrowly as raw-field codecs rather than as unrestricted tabular synthesizers. All four consume the same train, validation, and test partitions used by TraceCodec and are evaluated after PCAP rendering. Their decoded raw rows use the restricted renderer described below, whereas TraceCodec realizes PacketActions through its state-aware compiler.

The selected baselines span distinct tabular modeling choices: TVAE and TTVAE are explicit variational codecs, TabSyn-VAE uses the VAE backbone of a recent latent-diffusion tabular model, and GOGGLE adds a learned-relational generator outside that VAE family. This selection evaluates the raw-field interface across both variational and relational architectures.

All raw-field baselines are trained with validation-selected checkpoints under fixed optimizer-step budgets, rather than with a fixed number of epochs. This choice makes the comparison stable under

large raw-packet tables: an epoch over the CICIDS2017 Monday packet table contains millions of rows, so epoch counts are a dataset-size-dependent unit of computation. The validation objective is the model's raw-table reconstruction objective on the validation split, and the decoded test outputs are always produced from the best validation checkpoint. The step budgets and validation intervals are specified in Appendix F; the run manifests record the final step, best step, stopping reason, validation objective, and full training configuration.

Latent-interface controls. The latent-flow experiment in the main text uses only raw-field baselines whose latent code can be decoded without access to the original input row. TVAE satisfies this criterion directly: the decoder maps a row latent vector to the transformed raw packet table. TabSyn-VAE also satisfies it after flattening its non-CLS field-token latents; the flattened vector can be generated by the same latent-flow model and reshaped before the TabSyn decoder and field heads recover a raw row.

TTVAE is not used as a latent-interface control because its decoder is not a function of the latent vector alone. In the adapted implementation, the encoder first maps the transformed row to a Transformer encoder output $\mathbf{h}_{\text{enc}}$, then projects that same output to μ and $\log \sigma^2$. The decoder maps z to decoder tokens but reconstructs through a Transformer decoder that cross-attends to $\mathbf{h}_{\text{enc}}$. Thus reconstruction has the form $D(z, \mathbf{h}_{\text{enc}}(x))$, not $D(z)$. A downstream generator would therefore have to keep the input-derived encoder memory, generate a second high-dimensional memory sequence, or replace it with an arbitrary placeholder. Each choice changes the latent decode contract rather than testing whether a standalone raw-field latent sequence is useful.

GOGGLE is also excluded from the latent-flow control set. Although its model has a learned row-level latent, the validation-selected GOGGLE checkpoint produces no decoded rows satisfying the shared raw-packet table contract on CICIDS2017 Monday under the non-repair policy, as shown in Table 11. Running a downstream generator on that latent space would therefore mostly reproduce the same pre-render contract failure rather than test latent sequence modelability. We keep GOGGLE in the main decoded-PCAP comparison, where that failure mode is part of the raw-field baseline evidence.

C.3 Rendering policy

To keep the comparison fair, baseline outputs are rendered back into packets under a restricted deterministic policy. Decoded rows are split back along the original reference PCAP boundaries before rendering, matching the evaluation convention used for TraceCodec. Local recomputation is allowed only for quantities that are straightforward consequences of a single decoded packet instance, such as checksums or directly derived length fields. In practice, packet libraries may materialize these packet-local derived values during serialization; we treat that as ordinary rendering rather than as heuristic repair. Sequence-level or flow-state repair is not allowed. In particular, the baseline renderer may not impute missing active fields, infer hidden endpoints or transport state, repair decoded sequence or acknowledgment progression, or borrow TraceCodec's packet-action compiler logic. Rows that already violate the raw-table contract are counted as invalid decoded rows and are not repaired before rendering. This restriction is what makes the comparison meaningful: if raw-field baselines require stateful repair to become parseable, the repair has become part of the representation problem.

A protocol state machine is deterministic only once its input event is well specified. Independently decoded raw fields may fail to specify a unique legal event under the current flow state: flags, sequence progress, acknowledgment progress, endpoint orientation, timing, and payload length can disagree. A stateful renderer must then choose a precedence rule. If it trusts the raw fields, it may emit inconsistent packets; if it trusts protocol state and overwrites raw fields, it is no longer evaluating raw-field reconstruction; if it chooses case by case, it has become heuristic repair. Such a precedence rule is an implicit codec/compiler contract. The non-repair policy therefore isolates the raw-field interface rather than forbidding a stronger method: a raw baseline augmented with a full stateful compiler has moved toward the explicit interface design used by TraceCodec.

C.4 Raw-table diagnostics

Table 11 reports the pre-render raw-table diagnostics for the published baselines. These rows are deliberately not mixed into the main decoded-PCAP table: they are useful for explaining whether a

Table 11: Pre-render validity diagnostics for raw-field baselines.

Dataset	Method	Inv. row (%) ↓	Cat acc. (%) ↑	Num err. ↓
CICIDS2017 Monday	TVAE	37.46	71.68	4.12×10^{8}
	TabSyn-VAE	2.39	84.74	2.79×10^{7}
	GOGGLE	100.00	–	–
	TTVAE	37.28	65.99	2.57×10^{8}
MAWI 202004071400	TVAE	14.22	80.68	1.64×10^{8}
	TabSyn-VAE	28.06	87.36	1.46×10^{7}
	GOGGLE	100.00	–	–
	TTVAE	61.97	67.17	5.26×10^{7}

Table 12: Raw-table oracle roundtrip under the restricted renderer.

Dataset	TCP event dist. (pp) ↓	Count (%) ↓	Proto (%) ↓	IAT TV (%) ↓	Flow cnt. (%) ↓	Flow dur. TV (%) ↓
CICIDS2017 Monday	0.002	0.42	0.42	0.13	0.09	0.64
MAWI	0.063	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001

raw-field model fails before rendering, but they are undefined for TRACECODEC and the Oracle path because those methods do not decode through raw packet-table rows.

C.5 ORACLE ROUNDTRIP

We additionally run one no-learning oracle control:

$$\text{PCAP} \rightarrow \text{raw packet table} \rightarrow \widehat{\text{PCAP}}.$$

This is not a competing baseline. It simply checks whether the shared raw table and restricted renderer can already close a near-exact roundtrip when given ground-truth rows directly. We report the check on CICIDS2017 Monday and MAWI.

Its role is interpretive. If this oracle roundtrip is near-exact while learned raw-field baselines still decode poorly, the failure is not primarily in the single-packet contract or the renderer. It appears after learning, when a generic raw-field codec has to carry cross-packet behavior through its bottleneck without the state variables used by TRACECODEC.

D METRIC DEFINITIONS

This appendix defines the metrics reported in the main tables and figures. Percent columns multiply the underlying fraction or distance by 100. "pp" denotes percentage points. All decoded-PCAP metrics are computed after packet rendering, not on intermediate token sequences.

Shard-aligned decoded-PCAP evaluation. Unless otherwise stated, a decoded test split is compared against the matching reference shards $\{x_s\}_{s=1}^{S}$. Packet analyzers run on each reference/decoded shard pair and the resulting summaries are aggregated. This avoids mixing independent capture units and keeps sessionization, TCP analysis, and packet-count accounting aligned across methods.

TCP event dist. The main-table TCP event distance compares the rate profile of structured TCP diagnostic events. Let $\mathcal{A}$ contain retransmission, fast retransmission, spurious retransmission, lost segment, ACKed lost segment, duplicate ACK, out-of-order segment, and zero-window event. If $c_a(x)$ and $c_a(\hat{x})$ are the counts of event a in the reference and decoded PCAPs, and $N_{\text{tcp}}(x)$, $N_{\text{tcp}}(\hat{x})$ are their TCP packet counts, the table reports

$$\sum_{a\in\mathcal{A}} \left| \frac{c_a(\hat{x})}{\max(N_{\text{tcp}}(\hat{x}),1)} - \frac{c_a(x)}{\max(N_{\text{tcp}}(x),1)} \right|$$

in percentage points. This is a profile-fidelity metric: real captures may contain TCP diagnostic events, so the target is the reference profile rather than zero events.

Count. If N and $\hat{N}$ are the reference and decoded packet counts, the Count column reports

$$\frac{|\hat{N} - N|}{\max(N, 1)}.$$

This penalizes decoded traces that silently drop or invent packets. The latent-interface appendix labels the same quantity as Count err.

Proto. Let p and $\hat{p}$ be protocol-ratio vectors over TCP, UDP, ICMP, and other packets. The Proto column reports total variation,

$$\mathrm{TV}(p, \hat{p}) = \frac{1}{2} \sum_i |p_i - \hat{p}_i|.$$

IAT TV. Inter-arrival times are histogrammed using fixed millisecond bins shared by all methods. If h_{iat} and $\hat{h}_{\mathrm{iat}}$ are the normalized reference and decoded histograms, the IAT TV column reports $\mathrm{TV}(h_{\mathrm{iat}}, \hat{h}_{\mathrm{iat}})$.

Flow cnt. Flows are measured as bidirectional transport sessions under the same sessionization rule used for duration analysis. If M and $\hat{M}$ are the reference and decoded session counts, the Flow cnt. column reports

$$\frac{|\hat{M} - M|}{\max(M, 1)}.$$

The value can exceed 100% when a method fragments traffic into many more sessions than the reference.

Flow dur. TV. Each bidirectional five-tuple is split into transport sessions before duration measurement. TCP sessions end at RST/FIN boundaries or after a 60-second idle gap; UDP, ICMP, and other sessions use the same idle split. Session durations are histogrammed using fixed millisecond bins, and Flow dur. TV reports total variation between the reference and decoded duration histograms.

TCP transition distance. Figure 4 compares event-to-event TCP trajectories within bidirectional sessions. Each trace is converted into a row-normalized transition matrix over coarse TCP event labels. The scalar transition distance reported in the text is the weighted row-wise total variation between the decoded and reference matrices, with row weights taken from the reference transition mass.

Flow-switch and run-length diagnostics. Figure 5 measures whether decoded traces preserve multi-flow interleaving. The packet-to-packet flow-switch rate is the fraction of adjacent decoded packets whose bidirectional session label changes. Same-flow run length is the length of a maximal consecutive packet run belonging to the same bidirectional session. Session count is the same sessionization count used by Flow cnt.

Action field acc. The ablation-table Action field acc. column is the mean PacketAction field accuracy for TRACECODEC on the held-out split. It averages exact-match accuracy over decoded PacketAction fields under the active masks of the schema, so inactive protocol-specific fields do not contribute. This is an intrinsic codec diagnostic; decoded-PCAP metrics remain the primary fidelity evidence.

KL rate. The ablation-table KL rate reports the test posterior KL of the codec encoder to the unit-Gaussian prior,

$$\mathbb{E}\left[\sum_{k=1}^{d} \frac{1}{2}\left(\mu_k^2 + \exp(\log \sigma_k^2) - 1 - \log \sigma_k^2\right)\right],$$

in nats per packet over latent dimension d. We report the raw posterior KL without free-nats clamping. The metric is an intrinsic latent-regularity diagnostic: removing KL can leave reconstruction metrics nearly unchanged while moving the posterior much farther from the unit-Gaussian prior. We do not train a separate downstream generator for each ablation.

Usable suffixes. In the latent-interface experiment, each case asks a downstream generator to produce a 1024-token suffix from a matched 1024-token prefix. Usable suffixes is the fraction of decoded cases whose rendered PCAP contains exactly the requested suffix packet count. It is a case-level measure of whether the sampled latent suffix remains on a complete decode surface.

Valid pkt coverage. Valid packet coverage is the decoded valid packet mass divided by the requested suffix packet mass, summarized over latent-interface cases. It gives partial credit when a suffix decodes to some valid packets but loses or invalidates the rest.

Coverage-adjusted TV. Latent-interface distributional metrics compare decoded suffixes with their matched held-out real suffixes. Let P_b and Q_b be the normalized reference and valid-decoded histograms. Let N be the reference packet count, M the decoded packet count before validity filtering, and V the valid decoded packet count. We define

$$\text{CA-TV}(P, Q) = \frac{1}{2}\left[\sum_b \left|P_b - \frac{V}{N}Q_b\right| + \frac{(M-V)_+}{N} + \frac{|N-M|}{N}\right],$$

with $Q_b = 0$ when $V = 0$. The first term measures distribution mismatch at the surviving mass; the remaining terms charge invalid packets and packet-count mismatch. When $M = V = N$, the metric reduces to ordinary TV. The CA IAT TV, CA pkt-size TV, CA flow-pkts TV, and CA sess-dur TV columns apply this accounting to inter-arrival-time, packet-size, packets-per-flow, and session-duration histograms, respectively.

Raw-table diagnostics. The auxiliary raw-table table reports pre-render diagnostics for raw-field baselines only. Inv. row is the fraction of decoded rows that violate the shared raw-packet contract before packet rendering. Cat acc. is mean exact-match accuracy over active categorical raw-table columns on rows that survive the invalid-row filter. Num err. is mean absolute error over active numerical raw-table columns in raw field units. These diagnostics explain pre-render raw-table behavior and are not mixed into the main decoded-PCAP comparison.

E Extended Results

This appendix reports a paired comparison of deterministic and learned realization, followed by additional results on MAWI.

E.1 Replacing deterministic compilation with a learned realizer

We compare deterministic and learned realization in a paired 2×2 experiment. For each dataset, the action source is either the ground-truth PacketAction sequence or one reconstruction sampled from the trained codec and then frozen. Each sequence is realized by either the deterministic compiler or a learned causal model. The paired paths therefore receive identical actions, timing, packet order, and PCAP boundaries. Because each decoded source is a single stochastic reconstruction, compiler values in this table need not match values from a different reconstruction run in Table 3; only within-source comparisons are paired.

The learned replacement is a one-layer causal LSTM with hidden width 768 and 4,272,965 trainable parameters. It receives all 36 PacketAction fields and δ_t, then predicts 22 complete raw packet-table fields. Its training objective covers all source-PCAP targets, including endpoint literals, ports, and absolute sequence and acknowledgment values. PacketActions deliberately omit some of these source-specific values, so they are not uniquely determined by the input action sequence. We therefore evaluate the structure of the rendered PCAP rather than exact source-field recovery and treat the experiment as a comparison with this particular full-field learned replacement. We train a separate realizer for each dataset on 256-packet segments for three epochs with batch size 32 and AdamW at learning rate 0.001. Epoch 3 is selected by validation loss. The causal hidden state crosses inference chunks and resets only at PCAP boundaries.

The learned path does not call the FlowTable, TCP state machines, compiler constraints, or sequence repair. Its packet-local serializer only selects protocol-specific fields, accumulates predicted timing,

Table 13: Paired realization comparison across three datasets. Within each action-source pair, the compiler and LSTM receive identical actions, timing, packet order, and PCAP boundaries. Bold marks the better result within a pair; ties are unbolded. TCP distance is in percentage points; all other metrics are percentages. Lower is better except for valid coverage and IP parseability.

Dataset	Action source	Realizer	TCP event dist.	Count	Proto	IAT TV	Flow cnt.	Flow dur. TV	Valid cov.	IP parse
CICIDS2017	Ground truth	Compiler	**2.17**	**0.00**	**0.00**	**0.04**	**0.00**	**0.01**	**100.00**	100.00
CICIDS2017	Ground truth	Causal LSTM	43.04	2.02	0.76	86.64	694.35	77.30	97.98	100.00
CICIDS2017	Decoded	Compiler	**12.66**	**0.00**	**0.00**	**0.49**	**0.04**	**9.13**	**100.00**	100.00
CICIDS2017	Decoded	Causal LSTM	43.52	10.25	3.95	85.89	631.22	77.61	89.75	100.00
MAWI	Ground truth	Compiler	**1.55**	**0.00**	**0.00**	**0.01**	**0.00**	**0.00**	**100.00**	100.00
MAWI	Ground truth	Causal LSTM	74.19	0.04	0.02	85.89	51.20	14.36	99.96	100.00
MAWI	Decoded	Compiler	**2.32**	**0.00**	**0.00**	**0.39**	**0.07**	**0.25**	100.00	100.00
MAWI	Decoded	Causal LSTM	54.86	0.00	22.51	79.71	59.38	16.40	100.00	100.00
VNAT	Ground truth	Compiler	**0.00**	**0.00**	**0.00**	**0.02**	**0.00**	**0.00**	**100.00**	100.00
VNAT	Ground truth	Causal LSTM	20.11	28.51	22.69	92.68	28,628.51	69.46	71.49	100.00
VNAT	Decoded	Compiler	**0.14**	**0.00**	**0.01**	**1.09**	**7.46**	**6.32**	**100.00**	100.00
VNAT	Decoded	Causal LSTM	0.67	100.00	28.03	98.59	97.33	76.28	0.002	100.00

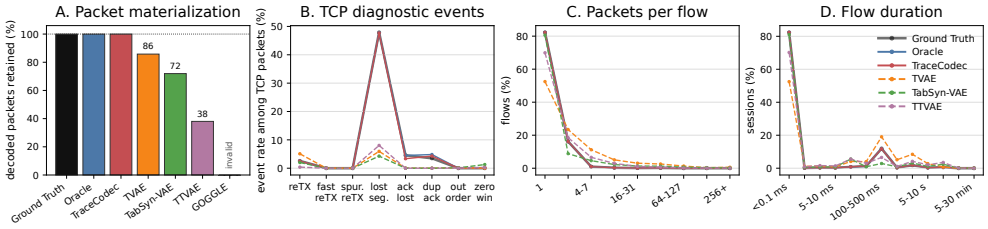


Figure 6: Protocol-state diagnostics on MAWI, using the panels from Figure 3.

computes lengths and checksums, and writes PCAP bytes. Thus, it can materialize a predicted row but cannot repair flow identity or transport state across packets.

Table 13 compares the two realization paths while holding each action source fixed. On CICIDS2017, replacing the compiler with the LSTM raises TCP-event distance from 12.66 to 43.52 percentage points and flow-count error from 0.04% to 631.22% for the same decoded actions. On MAWI, the replacement raises TCP-event distance from 2.32 to 54.86 percentage points and IAT TV from 0.39% to 79.71%. On VNAT, only 0.002% of the LSTM output from decoded actions satisfies the shared packet contract, compared with 100% for compiler output. Its low TCP-event distance is therefore not evidence of faithful realization because almost no predicted rows survive the packet contract.

All compiler outputs have 100% valid coverage. Every LSTM output remains 100% IP-parseable, but its failures differ by dataset, including flow fragmentation, timing distortion, and invalid packet records. Parseability alone therefore does not preserve sessions or transport behavior. This single-seed control does not show that neural realizers can never learn protocol state. It shows that this causal learned replacement does not match deterministic compilation under the same action interface.

E.2 Additional MAWI results

The results below use the MAWI `202004071400` samplepoint-F trace and the temporal split described in Appendix A: 2,400 train shards, 300 validation shards, and 300 test shards, corresponding to 39.23M/1.70M/1.46M PacketAction tokens after the shared packet-level scope.

Figures 6–8 apply the decoded-PCAP and structural diagnostics from the main text to the MAWI held-out split. TraceCodec preserves packet materialization, TCP-event structure, and multi-flow progression more consistently than the raw-field baselines, which either lose packet mass or fragment transport and session structure after rendering.

Table 14 shows the same interface roles as the main ablation study. Coarsening timing raises IAT TV from 0.38% to 8.36%, while flow-count error remains at 0.07%. Removing TCP progression and window cues leaves action accuracy high but increases TCP-event distance from 2.31 to 115.29 percentage points. Removing KL keeps the decoded reconstruction metrics close but raises the test

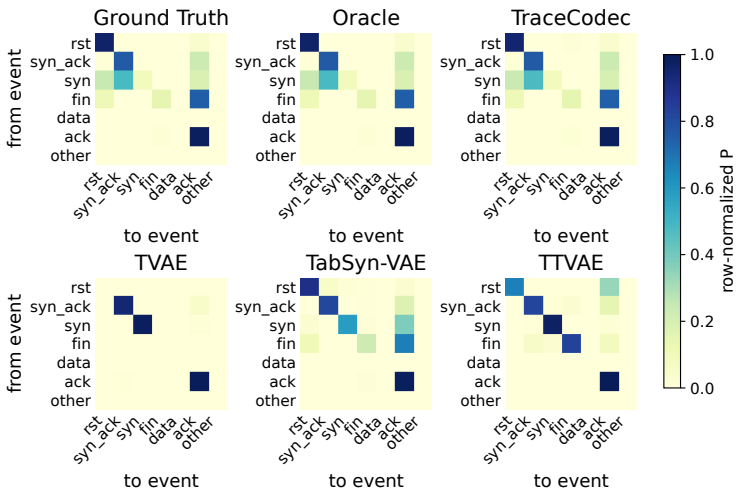


Figure 7: TCP transition consistency on MAWI. Each heatmap is a row-normalized transition matrix over coarse TCP event labels within bidirectional sessions, matching Figure 4.

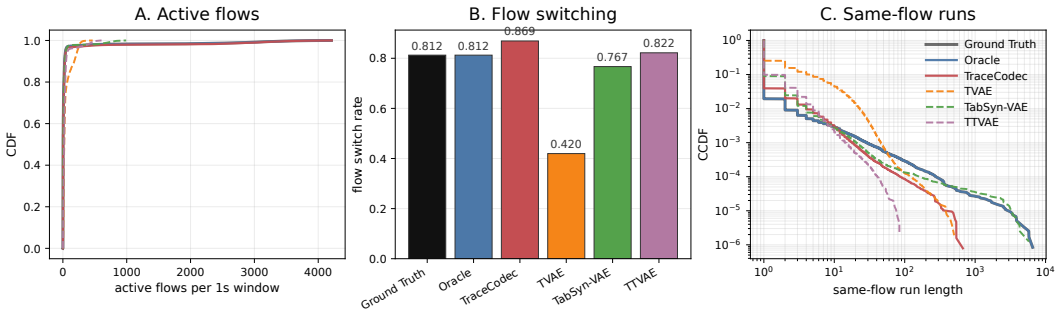


Figure 8: Active-flow and multi-flow interleaving on MAWI, using the diagnostics from Figure 5.

posterior KL rate from 249.38 to 5829.68 nats per packet. As on CICIDS2017 Monday, reconstruction fidelity and latent regularity are distinct properties.

E.3 Latent-interface experiment

The main text reports only a compact latent-interface check. This appendix gives the details needed to interpret that table. Each PCAP shard is first encoded into a packet-ordered latent sequence. Fixed prefix/suffix windows are then extracted with a 1024-token prefix, a 1024-token generated suffix, and stride 512. The setting is long enough to require continuation of timing and multi-flow interaction structure, but still small enough to train the same downstream model across representations.

The latent generator is a Transformer velocity model trained with conditional flow matching. It attends over the concatenation of prefix latents and noisy suffix latents, receives a scalar flow-time embedding, and predicts the suffix velocity. The reported latent objective is validation flow-matching MSE under fixed validation noise and time draws. For decoded evidence, sampled suffix latents are decoded back through the corresponding representation and compared against real held-out suffix PCAPs materialized from the same continuation windows, using the same decoded-PCAP diagnostics reported elsewhere in the paper. This evaluation asks whether a generator can stay on a usable decode surface while matching the actual continuation trace, rather than looking good only relative to a representation-specific oracle decode.

Table 14: MAWI ablation results for TRACECODEC. Boxes mark targeted metrics; KL rate is the encoder posterior KL to the unit-Gaussian prior, summed over latent dimensions and averaged per packet.

Variant	Action field acc. (%) ↑	TCP event dist. (pp) ↓	IAT TV (%) ↓	Flow count err. (%) ↓	Flow duration TV (%) ↓	KL rate (nats/pkt)
Full TRACECODEC	98.75	2.31	0.38	0.07	0.25	249.38
W/O KL	99.50	3.33	0.34	0.04	0.17	5829.68
W/O fine timing	99.05	2.81	8.36	0.07	0.41	141.66
W/O TCP cues	99.67	115.29	0.32	0.04	0.19	71.40

Table 15: Latent-interface check on MAWI. CA: coverage-adjusted total variation.

Representation	Usable suffixes (%) ↑	Valid pkt coverage (%) ↑	CA IAT TV (%) ↓	CA pkt-size TV (%) ↓	CA flow-pkts TV (%) ↓	CA sess-dur TV (%) ↓
TRACECODEC	**98.44**	**100.00**	**5.38**	**1.42**	**4.33**	**5.12**
TVAE	7.03	85.69	40.22	16.60	39.21	41.04
TabSyn-VAE	10.94	62.30	43.33	40.38	45.22	46.95

For TRACECODEC, each sequence element concatenates the normalized codec latent with a one-dimensional transformed time delta; the downstream model generates both components. The raw-field controls use their standalone latent codes, whose decoders reconstruct timing as a raw field.

For TRACECODEC, sampled latents are lowered through the fixed packet-action decoder and compiler. Suffix decoding starts from an empty runtime and replays the decoded within-window prefix, without emitting it, to rebuild flow and transport state before emitting the sampled suffix. For raw-field controls, sampled TVAE and TabSyn-VAE latents are decoded through their own raw-table decoders and then rendered under the same non-repair policy used in the main comparison. TTVAE and GOGGLE are not included in this latent-interface control set for the reasons described in Appendix C: TTVAE's decoder requires input-derived encoder memory in addition to z, and GOGGLE already fails the raw-row contract before rendering on CICIDS2017 Monday.

Table 15 repeats the same continuation protocol on MAWI. TRACECODEC again provides the most reliable decode surface, with nearly all suffixes usable, full median valid-packet coverage, and substantially lower coverage-adjusted timing, packet-size, flow-size, and session-duration distances than the raw-field latent controls. TVAE and TabSyn-VAE can produce parseable packets, but their sampled suffixes lose packet mass on many windows; the coverage-adjusted diagnostics therefore expose larger continuation errors than ordinary distribution distances alone.

The main ablation table uses the intrinsic posterior KL rate defined in Appendix D to measure latent regularity. The separate flow-matching experiment evaluates whether the resulting generator-facing representation supports traffic continuation.

F REPRODUCIBILITY DETAILS

This appendix records the public entry points and run metadata needed to reproduce the main result tables. The implementation writes JSON manifests at every dataset-build, training, reconstruction, and metric-aggregation stage, so each reported row can be traced back to the corresponding run output.

Software environment. All experiments are run from the repository root in a Python 3.10+ environment created for the project. A typical setup is:

```
conda create -n trace_codec python=3.10
conda activate trace_codec
python -m pip install --upgrade pip
python -m pip install -e .
python -m pip install jupyter ipykernel nbclient nbformat
python -m ipykernel install --user --name trace_codec \
  --display-name "trace_codec"
```

The package dependencies are declared in `pyproject.toml`; the extra notebook packages are only needed to execute the paper notebooks. Decoded-PCAP diagnostics use Wireshark's command-line tools, so `tshark` must also be available on `PATH`. For example, on Ubuntu:

```
sudo apt-get update
sudo apt-get install -y tshark
```

The paper runs used CUDA-enabled PyTorch on NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs with 96GB memory. The scripts expose `-device cuda` and can be restricted to a particular GPU with `CUDA_VISIBLE_DEVICES`. Runtime depends on the selected stages; the training budgets and evaluation caps are specified below and recorded in run manifests.

Notebook entry points. The main CICIDS2017 Monday workflow is launched from `trace_codec_experiment.ipynb`; the MAWI and VNAT workflows use `trace_codec_experiment_mawi.ipynb` and `trace_codec_experiment_vnat.ipynb`, respectively. After activating the environment above, the notebooks can be run interactively with the `trace_codec` kernel or executed unattended:

```
python -u scripts/execute_notebook_with_checkpoints.py \
  trace_codec_experiment.ipynb \
  --output outputs/trace_codec_experiment.executed.ipynb \
  --kernel-name trace_codec \
  --timeout -1 \
  2>&1 | tee outputs/trace_codec_experiment.nbconvert.log
```

Replace the notebook path and output log with the MAWI or VNAT names to reproduce those runs. The notebooks call the scripts listed below and skip expensive stages when their sentinel artifacts already exist.

Shared split and dataset construction. Within each dataset, all learned methods consume the same train, validation, and test partitions described in Appendix A. For the raw-field baselines, the shared packet table is built once and then reused by TVAE and TTVAE; TabSyn-VAE and GOGGLE derive their prepared numerical/categorical arrays from that same table:

```
bash scripts/run_packet_table_baseline_experiments.sh \
  --dataset-key cic_ids2017_monday \
  --split-root dataset/trace_codec_experiments/cic_ids2017_monday/split \
  --device cuda \
  --baselines tvae,tabsyn_vae,goggle,ttvae
```

The command can be restricted to individual stages with `-phases build`, `-phases train`, or `-phases reconstruct`. Reconstructions optionally include `tshark` expert-analysis diagnostics with `-tshark-expert`.

Raw-field baseline training protocol. Raw-field baselines use validation-selected checkpoints under fixed optimizer-step budgets. This avoids treating one pass over a multi-million-row packet table as the primitive unit of comparison. The default budgets are TVAE: 100k optimizer steps, TTVAE: 100k steps, GOGGLE: 100k steps, and TabSyn-VAE: 15k steps. TVAE, TTVAE, and GOGGLE are evaluated every 5k steps; TabSyn-VAE is evaluated every 1k steps. Training stops at the step budget or after eight consecutive non-improving validation events, with `min_delta=0`. The selected checkpoint minimizes the validation raw-table reconstruction objective, and all held-out reconstructions are decoded from that checkpoint.

The corresponding script options are:

```
--max-steps tvae=100000,tabsyn_vae=15000,goggle=100000,ttvae=100000
--eval-interval-steps tvae=5000,tabsyn_vae=1000,goggle=5000,ttvae=5000
--patience-evals 8
--min-delta 0.0
```

Each training manifest records the full configuration, `global_step`, `best_step`, `stop_reason`, `best_val_objective`, and the final held-out reconstruction diagnostics. The accompanying `history.jsonl` file records every validation event, including the training-window averages since the previous validation event. For GOGGLE, learned-graph thresholding is also keyed to optimizer step, and reconstruction materializes the graph at the selected checkpoint step.

TRACECODEC training and artifact mapping. TRACECODEC dataset construction, training, decode, and evaluation are launched from `trace_codec_experiment.ipynb`, which calls the repository scripts under the same Python environment. We train for at most 50,000 optimizer steps with AdamW, effective batch size 30,000, learning rate 10^{-4}, and a cosine schedule with 2,000 warmup steps. Training uses bfloat16, seed 42, KL weight 5×10^{-4} with a 16-epoch warmup, 0.01 free nats per latent dimension, and a timing-loss weight of 4.0 after clipping time deltas at 10 seconds. We validate every 1,000 steps and export the checkpoint with the lowest validation loss.

Generated outputs are written under `outputs/trace_codec_experiments/<dataset>/<variant>/`; run metadata and training curves are written under `runs/trace_codec_experiments/<dataset>/<variant>/`. The main comparison rows are summarized from the reconstructed test PCAP reports, while intrinsic codec diagnostics and latent checks are written under the corresponding variant output directories.

The posterior KL-rate diagnostic in the ablation table is computed directly from the exported codec checkpoint and test split:

```
OUT=outputs/trace_codec_experiments/cic_ids2017_monday/full
python scripts/compute_trace_codec_latent_rate.py \
  --model-dir runs/trace_codec_experiments/cic_ids2017_monday/full/model \
  --split test \
  --batch-size 30000 \
  --device cuda \
  --output "$OUT/latent_rate_test.json"
```

The script reads the dataset path and dataset-variant contract from the run artifacts, then reports raw posterior KL without free-nats clamping.

Latent-interface training protocol. The latent-interface experiment is launched by the same notebooks through `scripts/run_latent_sequence_experiments.sh`. For each representation, the script first exports shard-aligned latent sequences for the shared train/validation/test split. A TRACECODEC sequence element is $[\mu_t; y_t]$, where μ_t is the normalized codec latent and $y_t = \log(1 + \delta_t/\tau)$ is the explicit time coordinate using the codec's training-set time scale τ. The flow model predicts both components with objective $\mathcal{L}_\mu + 0.04\mathcal{L}_y$; generated timing is decoded from the predicted y_t, not from the VAE timing head. The script then trains a conditional latent-flow model on fixed prefix/suffix windows with stride 512. The paper-facing CICIDS2017 Monday setting uses a 1024-token prefix and a 1024-token generated suffix. Unless otherwise stated, the flow model is a 12-layer Transformer with hidden width 768, 12 attention heads, MLP ratio 4, dropout 0.05, logit-normal flow-time sampling, bfloat16 autocast, no activation checkpointing, and AdamW with learning rate 2×10^{-4}, weight decay 10^{-2}, gradient clipping at 1.0, 1000 warmup steps, cosine decay to a 0.2 learning-rate ratio, and EMA decay 0.9999. The completed run uses batch size 80, 32k optimizer steps, and validation/checkpointing every 500 steps. Checkpoints are selected by the lower validation flow-matching MSE between raw and EMA weights; manifests record the selected source, step, and full configuration. Evaluation samples suffixes with a 400-step Heun solver, reports latent metrics over up to 512 windows, decodes up to 128 suffix cases into PCAPs, and compares those predicted suffix PCAPs with real held-out suffix PCAPs materialized from the same continuation windows.

The corresponding command shape is:

```
bash scripts/run_latent_sequence_experiments.sh \
  --dataset-key cic_ids2017_monday \
  --split-root dataset/trace_codec_experiments/cic_ids2017_monday/split \
  --device cuda \
  --representations trace_codec,tvae,tabsyn_vae \
  --prefix-length 1024 \
```

```
  --horizon 1024 \
  --stride 512 \
  --train-batch-size 80 \
  --train-max-steps 32000 \
  --train-eval-steps 500 \
  --train-lr 2e-4 \
  --lr-scheduler cosine \
  --warmup-steps 1000 \
  --min-lr-ratio 0.2 \
  --ema-decay 0.9999 \
  --d-model 768 \
  --num-layers 12 \
  --num-heads 12 \
  --dropout 0.05 \
  --time-loss-weight 0.04 \
  --sample-steps 400 \
  --sample-solver heun \
  --max-metric-windows 512 \
  --max-decoded-cases 128 \
  --tshark-expert
```