Automotive Cause-Effect Chains in the Wild: A Real-World Industrial Benchmark and Generator

Silviu S. Craciunas*†, Matthias Becker‡, Paul Pop*,
*Technical University of Denmark, Kongens Lyngby, Denmark
†NXP Semiconductors, Vienna, Austria
‡KTH Royal Institute of Technology, Stockholm, Sweden
Emails: silviu.craciunas@nxp.com, mabecker@kth.se, paupo@dtu.dk

***Abstract*—Modern automotive systems integrate large numbers of mixed-criticality functions on shared multi-core multi-SoC platforms. Sensor-to-actuator data propagation in these systems is commonly modeled as cause-effect chains that are governed by maximum data-age or reaction-delay budgets. This paper presents the statistical fingerprint of a real-world Advanced Driver-Assistance System (ADAS) use case provided by TrustMotion and introduces a synthetic benchmark comprising 500 task-set instances, organized into five architectural-scale buckets. These instances are derived from the fingerprint by replicating the source compute platform and chain corpus at each scale. We document how the instances were produced, which real-world characteristics they preserve, which aspects they intentionally abstract away, and how researchers can use the data to evaluate mapping, scheduling, and/or end-to-end latency analyses on inputs whose statistical fingerprint reflects automotive reality.**

I. Introduction

Automotive computing platforms are moving away from hardware-centric designs built around many dedicated and distributed ECUs and toward centralized or zonal software-defined architectures. On these platforms, a growing number of heterogeneous mixed-criticality functions, related to Advanced Driver-Assistance Systems (ADAS), assisted-driving features, and infotainment services, are consolidated onto shared high-performance multi-core multi-SoC hardware [1], [2]. A representative driver-assistance pipeline acquires data from cameras and ranging sensors (ultrasonic, lidar, radar), processes it in sensor-fusion components, and forwards the result to planning, navigation, and control algorithms that ultimately issue actuator commands, e.g., an emergency-brake activation. The correctness of such a pipeline depends not only on each software component meeting its deadline, but also on the temporal consistency of the data as it gets processed through the entire sequence. These end-to-end dependencies are captured as *cause-effect chains*, which are ordered sequences of communicating tasks representing a sensing/processing/actuation pipeline. Cause-effect chains have become a central modelling primitive in automotive timing analysis [3], [4]. A cause-effect chain is a tuple $\zeta = (\langle \tau_1, \tau_2, \ldots, \tau_k \rangle, L, \pi)$ where the task sequence is ordered by data propagation, L is the end-to-end latency budget, and π is a chain-level importance weight. Tasks in a chain may run at different periods and on different cores. Under *reaction delay* the chain bounds the worst-case time from the earliest start of every instance of τ_1 to the latest end of τ_k along matched job instances. Under *data age*, it bounds the age of the data observed at every instance of the chain sink τ_k. Günzel et al. [3] define both semantics and the relationship between them. There have been many works dealing with analyzing and/or enforcing the budgets of cause-effect chains (cf. [4]–[8]).

Progress in the real-time community is sometimes slowed down due to the difficulty of obtaining publicly shareable data sets and problem instances that faithfully reflect industrial reality. Kramer et al. [9] observed that the absence of such benchmarks creates a real risk of divergence between academic research and industrial practice, and proposed a methodology in which aggregate characteristics of real systems are published in place of the raw data so that researchers can validate their solutions on industrially representative input. Concurrent work by Denzinger et al. [10] takes the same fingerprint-style approach on a CARIAD motion-and-drive ECU and ships the *Driverator* framework, where AUTOSAR-style task sets are reconstructed from statistical abstractions of Safety Integrity Levels, basic-software overheads, and memory constraints (Weibull distributions plus per-subject correlation coefficients) while preserving the source IP. We take a similar indirection via statistical-fingerprinting and describe a real automotive use-case specifically detailing the cause-effect-chain level. Furthermore, we also release the generator that can generate based on such fingerprints faithful synthetic use-cases scaled to different sizes within a user-specified bounded deviation. We also release synthetic *instances* (500 task-set files, organized in five architectural-scale buckets) generated from that statistical fingerprint using our generator. We hope that in the future we can release more fingerprints of real-world use-cases. Even without the generator, we describe the source use-case in anonymized form so that researchers can produce real-world inspired benchmarks of their own[1].

The remainder of the paper introduces the source use-case together with its task and chain characteristics (Section II), and describes the statistical fingerprint, the synthesis pipeline, and the format and observed properties of the released instances (Section III). We conclude the paper in Section IV.

[1]The released benchmark and generator, which contains also the statistical fingerprint of a real-world automotive ADAS controller and 500 pre-generated task-set instances synthesised from that fingerprint, can be found at https://github.com/dtu-ese/automotive-cause-effect-chain-benchmark

II. The Real-World Use-Case

The benchmark is anchored in a real-world automotive use-case integrated into millions of vehicles which we call the TrustMotion[2] benchmark. Throughout this paper we focus on this strictly computational view. The in-vehicle communication backbone (via, e.g., PCIe or TSN), while industrially important, is outside the scope of the released benchmark. We describe the use-case in three layers (platform anatomy, task model, and chain structure) and conclude with the statistical fingerprint in the spirit of Kramer *et al.* [9], so that researchers can calibrate or refit a generator of their own to obtain faithful synthetic instances.

A. Functional anatomy

The controller integrates three compute processors, each hosting tasks of a distinct functional role:

- **Decision-Making (`DM`).** Trajectory and speed planning. 3 cores.
- **Sensor Fusion (`SF`).** Fuses the upstream sensing streams (radar, lidar, ultrasonic, camera) into the decision-making input. 2 cores.
- **Time-Critical (`TC`).** Front-camera and driver-monitoring image processing. 4 cores.

Although in the original system we use Time-triggered (TT) and Fixed-priority (FP) scheduling, the released benchmark is policy-agnostic and researchers may analyze or schedule the use-case under any policy (FP, EDF, TT) and using any communication semantics (implicit, LET, etc.) as well as partitioned or global approaches. Cause-effect chains in the source either originate at an incoming bus message (typical sources include radar, lidar, ultrasonic, and camera signals) or at an internal sensor-side runnable, and terminate either by emitting an outgoing bus message or by handing data to an internal application task feeding a downstream subsystem. The dataflow direction is roughly: bus or sensor input $\rightarrow$ time-critical or fusion processing $\rightarrow$ decision-making $\rightarrow$ actuator-side bus output or internal handoff.

B. Task model

The source workload contains 102 periodic tasks distributed across the three processors ($62/24/16$ on `DM`/`SF`/`TC`). Each task carries the standard periodic-task parameters: period T, deadline D, worst-case execution time WCET, release time r, and an optional jitter constraint J. The jitter J can have different semantics, most commonly it bounds the start jitter of a task in each period instance. Other times, like for the TT approach in [2] J_i bounds, for every pair of consecutive instances of task τ_i, the deviation of the TT slot placement from being exactly one period T_i apart. Every task is bound to a specific CPU (`DM`, `SF`, `TC`) but within the CPU each task is either *pinned* (pre-assigned to a core) or *flex* (the OS or system designer is free to allocate it to any core).

[2]TrustMotion (formerly TTTech Auto), part of NXP, is a leading software platform product and service provider with a focus on System, Safety, and Security for the Software-Defined Vehicle (4SDV). https://www.trustmotion.com/

Execution-time information follows the WATERS convention [9]. Every task carries a per-instance worst-case WCET and a per-instance best-case BCET, with the sentinel $\mathrm{BCET} = 0$ reserved for tasks whose source did not record a separate best-case value. The TrustMotion source records only a single execution-time value per task so we set $\mathrm{BCET} = 0$. Fingerprints whose source includes a real BCET measurement (such as the WATERS-Bosch fingerprint) carry it through unchanged. Three further optional execution-time fields are admitted by the schema and propagated by the generator: (i) $\mathrm{WCET}_{\min}$ lower bound (the lower bound of a per-slot WCET envelope); (ii) an ACET (per-instance average-case time); and (iii) a per-instance execution-time distribution shape (Weibull α, β or empirical distribution). The $\mathrm{WCET}_{\min}$ is present in 8 of the 102 TrustMotion tasks where the communication-stack and per-host middleware runnables have execution time that is driven by the per-slot frame load on the inter-host bus giving a WCET range $[\mathrm{WCET}_{\min}, \mathrm{WCET}]$ reflecting the empirical envelope of the per-slot workload. The underlying parametric form of the load model (frame count, frame size) is a property of the bus, schedule, and the host CPU, all of which the benchmark abstracts away, so we expose only the envelope.

Across the 102 source tasks, some declare a non-zero earliest activation (EA) and/or a non-zero release jitter bound. The fingerprint captures EA and jitter at the distribution level rather than per task. For each it records the share of tasks that declare a non-zero value at all $53\,\%$ for EA, $11\,\%$ for jitter. Active EA and J values are binned as a fraction of the task period into a 7-bin histogram, so the distribution shape carries across periods. Note that if the source has many distinct active values spread over a range then binned histograms are created. If the source has only a handful of distinct active values (like in this case the jitter), the fingerprint lists them verbatim with per-100 shares. Hence active jitter values are not binned and the source uses only two discrete ratios, $J/T \in \{0.1, 0.5\}$ (9 and 2 tasks respectively).

C. Cause-effect chains

The use-case contains 39 budget-constrained cause-effect chains. Each chain has an ASIL label, an integer end-to-end latency budget, a priority weight in $[0, 1]$, and an ordered list of runnable references. There are interesting structural patterns shared by some of the chains. On one hand after the receive event there are middleware scheduling-manager and receive-side communication-stack pair (generic plus ASIL-specific) runnables and the symmetric transmit-side comm-stack pair and the middleware again on the send side. The middleware runnable therefore appear *twice* per chain, with 29 of the 39 source chains ($74\,\%$) containing at least one repeated task reference. The comm-stack and middleware tasks invoked on the receive leg are the *same task* invoked on the transmit leg, producing a $\tau_i \prec \ldots \prec \tau_i$ pattern within the chain. Moreover, some in-chain task repetitions occur, encoding the *cyclic stateful pass-through*. Here the chain lists the same runnable at *adjacent* positions $(k, k+1)$, sometimes with multiple successive replicas of the same task.

One example is ultrasonic preprocessing or object fusion, where chains may contain repeated runnable references of the form $\underbrace{\tau_{\mathrm{US}} \to \cdots \to \tau_{\mathrm{US}}}_{n \text{ periods}} \to \underbrace{\tau_{\mathrm{OBF}} \to \tau_{\mathrm{OBF}}}_{m \text{ periods}} \to \cdots$ where τ_{US} is the ultrasonic preprocessing task and τ_{OBF} is the object-fusion main runnable. Both idioms coexist in the same released fingerprint format where the chain is in either case an ordered task sequence with possible repeats and the generator preserves their marginal frequency.

ASIL distribution across the 39 chains is 27 ASIL-B, 4 ASIL-C, 8 unconstrained. No chain carries ASIL-D, although ASIL-D output runnables do appear. 26 of 39 chains begin with a `PDU_IN`, while the remaining start with an internal sensor-side runnable (camera, ultrasonic, etc). 35 of 39 chains end with a `PDU_OUT` and the remaining hand data to an internal application task that feeds a downstream subsystem.

III. Statistical fingerprint

We give the source fingerprint as a set of compact per-dimension tables in the WATERS 2015 style [9], so that a re-implementor can draw periods, WCETs, constraint flags, deadline ratios, chain shapes, and chain budgets independently from the tables that own those axes. All per-task statistics are reported on the 102-task pool.

Table I shows the platform skeleton (CPUs, cores, task counts, cumulative utilization per role). Table II gives the period histogram (102 tasks centred on 10–40 ms, with a 33.3 ms supervisory bucket and a 2.5 ms outlier from time-critical runnables). Table III gives the per-period WCET range and Table IV the per-CPU pinning prevalence (83 % overall pinning rate). Table V gives the D/T histogram (median 0.83 on the source pool, with 22.5 % implicit-deadline and 11 tasks at $D/T < 0.20$, including the per-core synchronisation tasks). The chain-length distribution is shown in Table VII (multi-modal, with clusters at $L \in \{9, 10\}$ and $L \in \{15, 16\}$). The period-transition bigram in Table VIII (with absorbing Start/End states where $116/466$ of all transitions stay at the 5 ms comm-stack inner walk).

Let $T_{\mathrm{th}} = \sum_{j=1}^{n} T_{\tau_j}$ denote the consumer-period sum of the chain over its n scheduled tasks (bus-PDU markers are excluded and the released JSON's chains are task-only, so this matches what the generator computes). Under implicit communication, Davare *et al.* [11] bound the chain's end-to-end latency by $\mathrm{Lat} \le \sum_i (T_{\tau_i} + R_{\tau_i})$. T_{th} is the period-only contribution to that bound, with PDU framing absorbed into the chain's external event boundary. The dimensionless tightness factor $\rho = b/T_{\mathrm{th}}$ records how aggressive the budget is relative to that period-only floor. The budget model is shown in Table VI (the empirical ρ histogram and the formula $b = \rho \cdot T_{\mathrm{th}}$). Across the 39 budgeted source chains, ρ has median 0.72, mean 1.00, $\mathrm{p}_{10} = 0.36$, $\mathrm{p}_{90} = 1.43$; 38 % of chains carry $\rho > 1$.

Together these tables are sufficient to drive a faithful generator, and to evaluate any implementation choice against the source dimension by dimension.

TABLE I: Source-side platform skeleton. $|\Gamma|$ is the number of tasks per CPU role. $\sum u$ is computed using $\max C$ for tasks with a recorded compute-time range, giving a per-role upper-bound utilization.

CPU	$\lvert\Gamma\rvert$	$\sum u$ (WCET)
`DM` (3 cores, FP)	62	2.84
`SF` (2 cores, TT)	24	1.81
`TC` (4 cores, TT)	16	3.25
total	102	7.90

TABLE II: Source-side period histogram per CPU role across the 102 tasks. The totals match the $|\Gamma|$ column of Table I. The distribution is centred on 10–40 ms.

T [ms]	`DM`	`SF`	`TC`	total	share
2.5	1	0	0	1	1.0 %
5	10	0	0	10	9.8 %
10	20	7	6	33	32.4 %
20	10	2	3	15	14.7 %
33.3	0	0	5	5	4.9 %
40	13	9	1	23	22.5 %
80	8	6	1	15	14.7 %
total	62	24	16	102	100 %

The generator is a direct replicator of the source fingerprint of Section II. For every released instance, all $100 \times k$ tasks and all chains are drawn fresh from the per-dimension distributions in a single anonymized `profile.json`.

A. The fingerprint profile

The fingerprint that drives the generator is captured in a single JSON file (`profile.json`) that contains *only* aggregate statistical distributions. Its top-level keys are:

- `platform` – CPU layout to be replicated.
- `chain_length_hist` – the $\{L \to n\}$ map over the 39 source chains (Table VII).
- `asil_distribution` – per-100-chain shares of ASIL labels ($B \approx 69.2$, $C \approx 10.3$, *none* ≈ 20.5).
- `budget_model` – the $\rho = b/T_{\mathrm{th}}$ decile-band representation (`rho_decile_edges`: 11 percentile edges $\{p_0, p_{10}, \ldots, p_{100}\}$) plus the formula $b = \rho \cdot \sum_{j=1}^{n} T_j$. No individual chain's ρ value is stored verbatim; the generator samples a decile uniformly and then a ρ uniformly within the decile.
- `sampling_fingerprint`:
  - `per_cpu_share` and `pinning_probability_per_cpu` – role-keyed (DM/SF/TC).
  - `period_histogram_global`, `period_histogram_per_cpu` (Table II).
  - `wcet_*_per_period_us` and `wcet_*_per_cpu_period_us`: per-(role, period) WCET data in one of three exchangeable forms. *Level A* is a pool of per-task WATERS-style records with mandatory wcet_max and bcet (with bcet $= 0$ as the sentinel for "the source did not measure it") plus optional wcet_min (lower bound of a per-slot WCET

TABLE III: Source-side per-period WCET envelope across the 102 tasks. Each row gives the bucket count and the $[\mathrm{WCET}_{\min}, \mathrm{WCET}_{\max}]$ range observed in the bucket. $\mathrm{WCET}_{\min}$ is the smallest observed WCET in that bucket (over the per-task single WCETs of ordinary tasks and over the per-slot WCET envelopes of the utility-runnable tasks).

T [ms]	n	$\mathrm{WCET}_{\min}$ [µs]	$\mathrm{WCET}_{\max}$ [µs]
2.5	1	100	100
5	10	80	1300
10	33	60	3250
20	15	90	3000
33.3	5	7000	22500
40	23	80	13500
80	15	100	40000

TABLE IV: Source-side per-task constraint prevalence across the 102 tasks. Per-role pinning probability is the per-task fraction with $\mathrm{pinned} = \mathrm{true}$.

constraint	count	share
earliest-activation > 0	54	52.9 %
declared jitter > 0	11	10.8 %
pinned on `DM`	56/62	90.3 %
pinned on `SF`	14/24	58.3 %
pinned on `TC`	15/16	93.8 %
pinned overall	85/102	83.3 %
constrained deadline ($D < T$)	79	77.5 %

envelope) and acet (per-instance average-case time). *Level B* is a binned multi-dimensional histogram in place of the pool, trading per-task identity for a stricter anonymity guarantee; this is the level used by the released TrustMotion fingerprint. *Level C* is the WATERS-style bucket aggregate (envelope plus optional Weibull or empirical distribution); the released WATERS-Bosch fingerprint uses it. The generator dispatches level A first, then B, then C.

- `wcet_range_per_period_us` – per-period envelope used as a clamp at sampling time (Table III).
- `deadline_ratio_histogram` (Table V), `ea_distribution`, `jitter_distribution` – binned D/T, r/T, and J/T ratios with an explicit "none" mass.
- `chain_bigram`, `chain_bigram_total`, `chain_last_period_distribution` – source period-transition counts (Table VIII).
- `chain_count_per_task`, `chain_same_task_repeat_share`, `pool_size_per_100_tasks`, `total_cores` – scalar generator targets.
- *optional WATERS extensions* – `activation_distribution_per_cpu` (per-role counts of *periodic* / *angle-sync* / *sporadic* tasks) and a per-record `distribution` key on the level-A / B / C WCET data (as `exec_distribution`).

The profile carries no source identifiers and no verbatim per-task or per-chain values. Every count is reported as a fraction per 100 tasks (or per 100 chains, or per 100 transitions for the chain bigram), pinning probabilities are rounded to a 0.05 granularity and the per-period WCET histograms use a fixed 50 µs bin grid. The ρ pool is reduced from 39 per-chain values to 11 percentile-decile edges. The released TrustMotion profile is ~ 24 kB.

TABLE V: Source-side deadline-to-period ratio D/T distribution across the 102 tasks. 22.5 % of tasks are implicit-deadline ($D = T$) and the bulk of the tasks sits in $[0.60, 1.00]$.

D/T bin	n	share
$[0.00, 0.20)$	11	10.8 %
$[0.20, 0.40)$	7	6.9 %
$[0.40, 0.60)$	11	10.8 %
$[0.60, 0.75)$	17	16.7 %
$[0.75, 0.90)$	13	12.7 %
$[0.90, 1.00)$	20	19.6 %
$D = T$	23	22.5 %
total	102	100 %

min 0.004, p_{10} 0.150, median 0.825, p_{90} 1.000, max 1.000. Mean 0.704.

TABLE VI: Source-side chain budget model. $T_{\mathrm{th}} = \sum_{j=1}^{n} T_j$ is the consumer-period sum of the chain. The tightness factor $\rho = b/T_{\mathrm{th}}$ is the single-parameter fingerprint stored.

$b = \rho \cdot T_{\mathrm{th}} = \rho \cdot \sum_{j=1}^{n} T_j$, grid = 5 ms		
ρ bin	n	share
$[0.20, 0.40)$	4	10.3 %
$[0.40, 0.60)$	4	10.3 %
$[0.60, 0.80)$	12	30.8 %
$[0.80, 1.00)$	3	7.7 %
$[1.00, 1.50)$	12	30.8 %
$[1.50, 2.00)$	0	0.0 %
$[2.00, 3.00)$	4	10.3 %
total	39	100 %

min 0.25, p_{10} 0.36, median 0.72, mean 1.00, p_{90} 1.43, max 2.97. The 15 chains with $\rho > 1$ are a mix of supervisory and long observer / state-machine chains.

B. Generation pipeline

We describe the generation pipeline to generate scaled versions based on the given fingerprint.

a) Platform.: For the platform, each bucket S_k has k replicas of the fingerprint three-CPU platform. We generate $3k$ CPUs with $9k$ cores total (`DM` contributes 3 cores per replica, `SF` 2, `TC` 4). CPU ids are numbered globally 0 to $3k - 1$.

b) Task sampling.: Each instance contains $100 \times k$ tasks drawn independently with the following per-task chain:

1) CPU role from $\mathrm{Multinomial}(\text{source per-CPU share})$;
2) period T from the period histogram;
3) WCET entry drawn from the source's per-(role, period) data, in whichever form the fingerprint provides (level A / B / C, see § III-A);
4) the released $\mathrm{WCET}' = \mathrm{round}(\mathrm{WCET} \cdot (1+\delta))$ with $\delta \sim \mathrm{Uniform}(-\varepsilon, +\varepsilon)$, clamped to the per-period envelope. When the fingerprint provides wcet_min or acet, those endpoints get independent jitter under the invariants $1 \le \mathrm{wcet\_min} \le \mathrm{wcet}$ and $\mathrm{bcet} \le \mathrm{acet} \le \mathrm{wcet\_min}$;

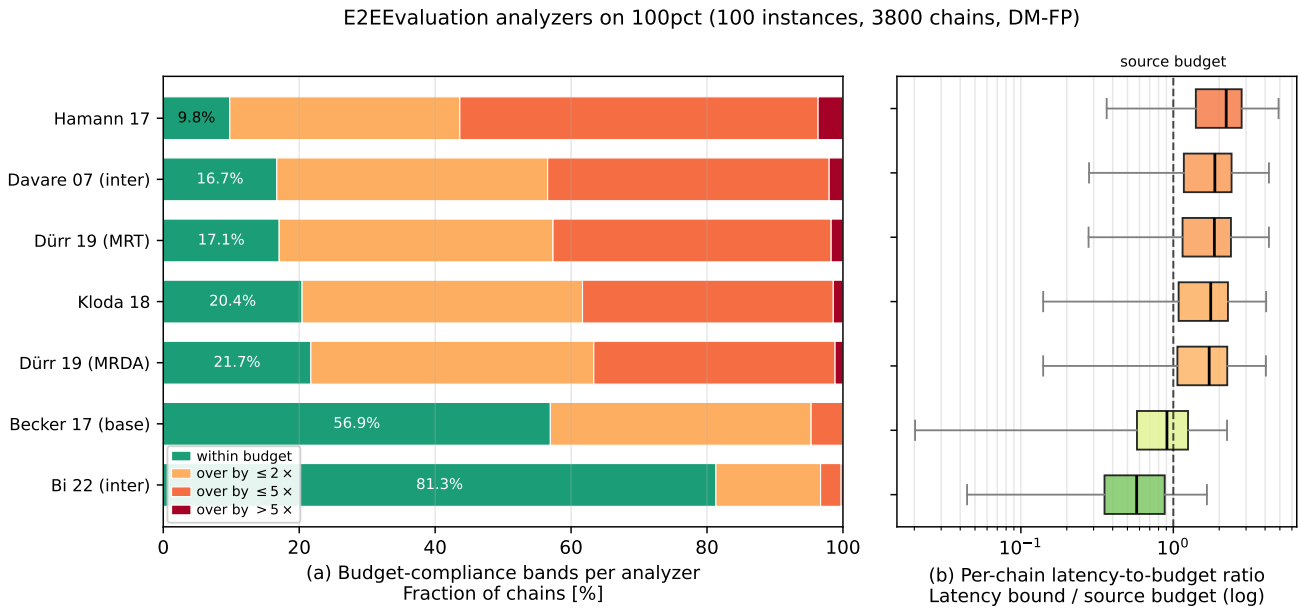

Fig. 1: End-to-end latency bound produced by seven public analyzers on the 100 instances of the `100pct` bucket (3 800 chains, deadline-monotonic fixed-priority scheduling). Panel (a) is a horizontal stacked bar per analyzer showing the fraction of chains in four budget-compliance bands – within budget (green), over by $\leq 2\times$, $\leq 5\times$, and $> 5\times$ (red). Panel (b) is a box plot of the per-chain ratio between the analyzer's bound and the chain's source budget; the vertical dashed line at 1 marks the budget, boxes coloured by within-budget fraction.

TABLE VII: Source-side chain length histogram (counts of compute runnables and `PDU_IN`/`PDU_OUT` markers).

L	n	share	L	n	share
2	1	2.6 %	11	2	5.1 %
5	3	7.7 %	12	2	5.1 %
6	2	5.1 %	13	1	2.6 %
7	1	2.6 %	14	1	2.6 %
8	2	5.1 %	15	6	15.4 %
9	6	15.4 %	16	4	10.3 %
10	6	15.4 %	17	2	5.1 %

total $= 39$, min $= 2$, mean $= 10.95$, max $= 17$.

5) deadline-ratio r from D/T histogram, $D = \mathrm{round}(r \cdot T)$, clamped to $[C, T]$;
6) CPU id picked among the k replicas of the chosen role, with the least-loaded one selected to keep per-CPU loads even. A boolean pinned flag is sampled per task from the fingerprint per-role pinning probability (Table IV) and recorded on the task.

After sampling, the generator runs a best-fit allocator that enforces a 5 % total-capacity safety margin $\sum u_i \leq 0.95 \cdot 9k$. The single knob ε is the global `--fingerprint-noise`, default 0.10, and bounds the per-task WCET jitter (and the optional wcet_min / acet jitter when the fingerprint provides them) as well as the per-chain budget jitter described below.

Chain end-to-end budgets in the source are not a smooth function of chain length, but they do admit a compact dimensionless fingerprint. At generation time, each synthetic chain draws a ρ uniformly from the source pool and applies $\pm\varepsilon$ multiplicative jitter on top:

$$b \;=\; \mathrm{round}_{5\,\mathrm{ms}}\big(\rho_{\mathrm{src}} \cdot (1+\delta) \cdot T_{\mathrm{th}}^{\mathrm{synth}}\big), \quad \delta \sim \mathrm{Uniform}(-\varepsilon, +\varepsilon) \tag{1}$$

where $T_{\mathrm{th}}^{\mathrm{synth}}$ is the consumer-period sum of the chain's actual sampled task sequence. If the resulting b would be smaller than $\sum_i C_i$ along the chain, the ρ draw is retried and the budget ultimately snaps to the next 5 ms boundary above the sum-WCET floor so every released chain is feasible by the trivial sum-WCET check.

Each instance contains $\mathrm{round}(100 \times k \times 0.382) \approx 38 \times k$ chains, that is 38/76/114/152/190 chains at S_1 through S_5. The generator builds every chain in three steps.

First, the chain length is set to $L = k \cdot L_{\mathrm{src}}$, where L_{src} is drawn from the source chain-length histogram in Table VII. Scaling the per-chain length together with the platform replication factor models the realistic case where a $k\times$ larger platform integrates $k\times$ more end-to-end functionality per chain rather than only $k\times$ more independent chains. Callers who prefer pure replication can opt out with `--length-scaling source`, which keeps $L = L_{\mathrm{src}}$ across all buckets.

Second, the period sequence $T_1, \ldots, T_L$ is obtained by walking the chain period bigram in Table VIII from the Start row for up to L steps. Dividing each cell by 466 gives its global transition frequency. During generation, outgoing transitions are normalized per source row to sample the next period in the bigram walk. The walk terminates earlier when the current bigram row offers only an End continuation, which preserves the source's natural termination semantics.

TABLE VIII: Source-side chain period-transition bigram with absorbing `Start` (S) and `End` (E) states. Each cell reports the count of from → to transitions out of the 466 total transitions across the 39 chains.

from\to [ms]	2.5	5	10	20	33.3	40	80	E
S	0	1	4	9	0	25	0	0
2.5	0	0	0	0	0	0	0	0
5	0	116	16	36	0	9	0	0
10	0	9	44	18	0	22	0	1
20	0	30	7	10	0	1	0	28
33.3	0	0	0	0	0	0	0	0
40	0	21	23	3	0	23	0	10
80	0	0	0	0	0	0	0	0

Rows for the 2.5, 33.3, and 80 ms buckets appear with all-zero transitions: those rates exist in the source workload (Table II) but no chain ever visits a task at any of them. Dividing each cell by 466 gives its global transition frequency in the source corpus. During generation, the current row non-End outgoing entries are used as relative weights for the next-period draw (End is not sampled as a successor) and the walk terminates early only when no non-terminal continuation is available.

TABLE IX: Released synthetic dataset (TrustMotion): per-bucket platform topology and task count. $|\Gamma|$ is the median per-instance task count ($100 \times k$ tasks per instance, i.e., 100 tasks per platform replica).

Bucket	Files	CPUs	Cores	$\|\Gamma\|$ (med.)	$\sum u$ (mean)	$\max u/\text{core}$
S_1	100	3	9	100	7.75	1.00
S_2	100	6	18	200	15.84	1.00
S_3	100	9	27	300	24.39	1.00
S_4	100	12	36	400	32.78	1.00
S_5	100	15	45	500	41.21	1.00

TABLE X: Released synthetic cause-effect chain characteristics (TrustMotion). Per-instance chain count is $\approx 38 \times k$ (matching the source's 39 chains scaled by k). Under the default `--length-scaling scale`, each chain's length is also multiplied by k (a $k\times$ larger platform integrates $k\times$ more end-to-end functionality per chain). Budgets are drawn from the source's empirical ρ pool (Table VI) with $\pm\varepsilon$ multiplicative jitter (Eq. (1)).

		Chain length			Budget [ms]	
Bucket	$\|\mathcal{L}_\aleph\|$	min	mean	max	min	max
S_1	38	2	10.7	17	7.5	837.5
S_2	76	4	21.5	34	15.0	1320.0
S_3	114	6	32.6	51	10.0	1845.0
S_4	152	8	43.6	68	25.0	2875.0
S_5	190	10	54.5	85	25.0	3160.0

Third, each period T_i is resolved to a concrete task drawn from the instance pool of tasks at period T_i, biased by the per-task visit-weight. Task identities remain unique within the chain. No repeats arise during the walk, which keeps the bigram-driven period sequence and the visit-weight bias intact.

A separate post-pass introduces same-task reuse. For each chain, the generator draws a Bernoulli trial with probability $p =$ `same_task_repeat_share`. On a hit, two distinct positions in $[0, L-1]$ are picked uniformly at random and the task at the first position is copied into the second. The source target $p = 0.744$ for the TrustMotion benchmark is matched within $\pm 1\,\%$ at every scale.

Finally, each chain is assigned an ASIL label sampled from the source's $\{B : 27, C : 4, \textit{none} : 8\}$ distribution and a budget per Eq. (1).

C. Released benchmark

Five scale buckets are released: $S_1, \ldots, S_5$ with $100, 200, 300, 400, 500$ tasks per instance respectively, each bucket containing 100 instances synthesized with different random seeds. Tables IX and X detail some aspects of the generated synthetic cases.

We then ran the seven analyzers on the released data under implicit communication, periodic activations, and fixed-priority scheduling on every chain of the 100-instance `100pct` bucket, i.e. 3 800 chains in total. For the evaluation we used the framework from [12][3]. We ran the baseline Davare et al. bound for distributed systems [11], the Becker et al. multi-rate effect-chain analysis without response-time information [4], [5], the Hamann et al. communication-centric bound [13], the chain-latency analysis of Kloda et al. [14], the maximum-reaction-time (MRT) and maximum-reduced-data-age (MRDA) variants of Dürr et al. [15], and the more recent Bi et al. analysis [16], which we run in its inter-chain form.

Fig. 1 shows the resulting per-chain latency distribution. The analyzers stratify in the expected order. Bi *inter* [16] is consistently the tightest, accepting $81.3\,\%$ of chains within budget. Becker 17 base [4], [5] sits in the middle at $56.9\,\%$. Dürr [15] reaches $21.7\,\%$ in its MRDA form and $17.1\,\%$ in its MRT form,

[3]available at https://github.com/tu-dortmund-ls12-rt/E2EEvaluation

Kloda 18 [14] reaches $20.4\,\%$, and Davare [11] reaches $16.7\,\%$. Hamann [13] is the loosest at $9.8\,\%$. Reported runtimes are sub-second per 38-chain instance for every analyzer except Kloda 18, which is sensitive to chain-shape pathologies in the synthetic data and takes the majority of the runtime.

D. What the benchmark abstracts away

First, the in-vehicle communication backbone is not modeled and hence cause-effect chains are pure sequences of compute tasks. Second, the synthetic schema does not declare per-core scheduling policy by default. Third, the TrustMotion source workload records only a single per-task WCET, so every released TrustMotion task ships with $\mathrm{bcet} = 0$ (the WATERS-style "not measured" sentinel). The schema reserves the `wcet_min` / `acet` / `exec_distribution` optional keys for fingerprints whose source includes those measurements (the second shipped fingerprint, WATERS-Bosch, is one such example), and an analyzer that wants a positive BCET hand-off can fall back to `wcet_min` (when present) or to `wcet`.

IV. Conclusion

The TrustMotion Cause-Effect Chain Benchmark is derived from a real integrated ADAS controller with 102 tasks on a three-CPU, nine-core platform and 39 ASIL-tagged cause-effect chains, of which 74% contain at least one repeated task reference. We have presented an anonymized statistical fingerprint of this use-case together with a pure-sampling generator and five scale buckets of synthetic instances (100 instances each, at 100 through 500 tasks per instance). Our hope is that the fingerprint and the generator can be used both by researchers to evaluate their methods and by industry practitioners to release their own use-cases and benchmarks in anonymized form without compromising IP.

Acknowledgments

This work has been funded by the Shift2SDV project (Grant Agreement No. 101194245), supported by the Chips Joint Undertaking and its members, including top-up funding by the national authorities of Austria, Denmark, Germany, Greece, Finland, Italy, Netherlands, Poland, Portugal, Spain, and Turkey.

The development of the software implementation and experimental (and plotting) infrastructure, as well as the writing of the paper, were assisted by Claude Opus 4.7 (Anthropic). All technical decisions, algorithm design choices, correctness validation, and scientific conclusions were made by the authors.

References

[1] G. Niedrist, "Deterministic architecture and middleware for domain control units and simplified integration process applied to ADAS," in *Fahrerassistenzsysteme 2016*. Springer Fachmedien Wiesbaden, 2018.

[2] S. D. McLean, S. S. Craciunas, E. Alexander Juul Hansen, and P. Pop, "Mapping and scheduling automotive applications on adas platforms using metaheuristics," in *Proc. ETFA*, 2020.

[3] M. Günzel, H. Teper, G. v. d. Brüggen, and J.-J. Chen, "End-to-end latency of cause-effect chains: A tutorial," *ACM Trans. Embed. Comput. Syst.*, vol. 24, no. 1, Dec. 2024.

[4] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *J. Syst. Archit.*, vol. 80, 2017.

[5] ——, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proc. RTCSA*, 2016.

[6] ——, "Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies," in *Proc. WATERS*, 2016.

[7] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich, "Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads," in *Proc. RTAS*, 2021.

[8] M. Günzel and M. Becker, "Optimal task phasing for end-to-end latency in harmonic and semi-harmonic automotive systems," in *Proc. RTAS*, 2025.

[9] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proc. WATERS*, 2015.

[10] T. Denzinger, M. Becker, and P. Ulbrich, "Shedding light onto safety integrity level and basic software constraints in a real-world automotive application: Case study with driverator framework," 2026. [Online]. Available: https://arxiv.org/abs/2605.04837

[11] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proc. DAC*. ACM, 2007.

[12] R. Edmaier, "Evaluation framework for end-to-end analysis," Master's thesis, TU Dortmund University, 2024. [Online]. Available: https://github.com/tu-dortmund-ls12-rt/E2EEvaluation/blob/main/doc/E2EEvaluation.pdf

[13] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication Centric Design in Complex Automotive Embedded Systems," in *Proc. ECRTS*, vol. 76, Dagstuhl, Germany, 2017.

[14] T. Kloda, A. Bertout, and Y. Sorel, "Latency analysis for data chains of real-time periodic tasks," in *Proc. ETFA*, 2018.

[15] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3358181

[16] R. Bi, X. Liu, J. Ren, P. Wang, H. Lv, and G. Tan, "Efficient maximum data age analysis for cause-effect chains in automotive systems," in *Proc. DAC*. ACM, 2022.