Compositional Analysis for Safety Specifications

Abhi Pomalapally and Arya Raeesi

University of California, Berkeley, Berkeley CA 94720, USA
{abhinav_pomalapally,aryaraeesi}@berkeley.edu

Abstract. Simulation-based testing is the dominant way to evaluate the safety of learning-enabled autonomous vehicles, but statistical model checking (SMC) scales poorly with scenario length. Compositional analysis addresses this by decomposing long scenarios into primitives whose traces can be generated in parallel and stitched together via importance sampling. Existing compositional frameworks, however, are restricted to *Markovian* specifications, which exclude most realistic driving requirements such as "stop at most once at a tollgate" or "once the vehicle exceeds 15 m/s, it must never drop below 6.5 m/s." We extend compositional analysis to non-Markovian specifications by representing each property as a deterministic finite automaton (DFA) and augmenting each simulation state with the induced automaton state. We divide the idea into two stages. The frontend decomposes composite scenarios, defined in the Scenic modeling language, into core primitives. The backend, integrated with the VerifAI tool, then propagates DFA state distributions across these primitive boundaries, using kernel density estimation (KDE) importance sampling to correct for distribution mismatch. On the MetaDrive simulator and on a Scenic intersection benchmark, our framework recovers monolithic estimates to within statistical error while achieving a 4–7× wall-clock speedup.

1 Introduction

AI-based systems are increasingly deployed in safety-critical settings such as autonomous driving and cyber-physical systems. Providing formal guarantees on their behavior is difficult: they include opaque components such as neural network controllers and operate in high-dimensional stochastic environments, ruling out classical exhaustive verification. A common alternative is statistical model checking (SMC) (5), which estimates the probability that a specification holds by repeated simulation, and which underlies toolkits for AI-based systems such as VerifAI (1).

Applied monolithically, SMC requires re-simulating the full scenario for every query, which is expensive when scenarios are long or composed of many subscenarios. Yalcinkaya et al. (10) address this with a *compositional* approach: they simulate only *primitive* scenarios once and reuse those traces, via kernel density estimation and importance sampling, to estimate the satisfaction probability of arbitrary *composite* scenarios built from those primitives. However, their framework has two limitations that restrict its applicability. First, it supports only Markovian specifications, whereas many natural safety properties depend on the history of the trace. Second, it takes scenarios in a custom format rather than in standard scenario modeling languages, such as Scenic (3; 9; 2).

In this paper, **we build a compositional analysis framework for non-Markovian safety specifications and add a Scenic frontend.** We represent the specification as a deterministic finite automaton (DFA) and propagate a distribution over its states from one primitive scenario to the next, conditioning on acceptance at each step. As in the Markovian case, kernel density estimation and importance sampling correct for the distributional mismatch between the end of one primitive and the start of the next; branches of probabilistic composition operators (such as Scenic's `choose`) are combined as a weighted average over per-branch estimates. We evaluate our approach in the MetaDrive simulator (6) on a range of composite driving scenarios and find that compositional estimates closely match monolithic ground truth while delivering a 4–7× wall-clock speedup.

2 Problem Definition

Primitive and composite scenarios. Let $\mathcal{S}$ denote a finite set of *primitive scenarios*. Each primitive $s \in \mathcal{S}$ induces a Markov decision process $M_s = (X, A, P_s, x_0^s)$ with shared state space X and action space A,

transition kernel P_s, and initial state x_0^s. Under the system policy, M_s yields traces $\tau = (x_0, x_1, \ldots, x_T)$ with $x_{t+1} \sim P_s(\cdot \mid x_t)$, distributed according to some trace distribution $\mathcal{D}_s$. A *composite scenario* c is built from primitives via three composition operators, following the design of Scenic (4):

(i) sequential composition $c_1 \,;\, c_2$, which runs c_1 to its terminal state and then c_2 from that state;
(ii) probabilistic choice $\mathsf{choose}\{(c_1, w_1), \ldots, (c_k, w_k)\}$, which executes c_i with probability $w_i / \sum_j w_j$; and
(iii) shuffle $\mathsf{shuffle}\{c_1, \ldots, c_k\}$, which executes the c_i's in a uniformly random order.

A composite scenario c induces a trace distribution $\mathcal{D}_c$ defined inductively over these operators. In practice, c is supplied as a *Scenic program* (3; 9; 2). Scenic is a probabilistic scenario specification language for autonomous systems whose programs declaratively describe ego behavior, environment objects, and composition structure, and whose simulator-agnostic design supports CARLA, Webots, MetaDrive, and LGSVL backends. Scenic's do, do choose, and do shuffle clauses correspond directly to the three operators above, and a Scenic source file thus defines a composite c together with the per-primitive worlds in which its primitives execute.

Specifications and satisfaction probability. A *specification* φ is given by a deterministic finite automaton $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, F)$ together with a labeling function $L : X \to \Sigma$. A trace $\tau = (x_0, \ldots, x_T)$ satisfies φ, written $\tau \models \varphi$, iff the run $q_0, q_1, \ldots, q_T$ given by $q_{t+1} = \delta(q_t, L(x_{t+1}))$ ends in an accepting state, $q_T \in F$. The quantity of interest is the *satisfaction probability*

$$\rho(c, \varphi) \;=\; \Pr_{\tau \sim \mathcal{D}_c}[\tau \models \varphi].$$

We call φ *Markovian* if $\tau \models \varphi$ depends only on the terminal state x_T, and *non-Markovian* otherwise.

The Compositional Non-Markovian Analysis Problem. Given a composite scenario c over primitives $\mathcal{S}$, a non-Markovian specification φ given by $(\mathcal{A}_\varphi, L)$, and a per-primitive simulation budget of N traces, produce an estimate $\hat{\rho}(c, \varphi)$ of $\rho(c, \varphi)$ together with a confidence half-width $\hat{\varepsilon}$ at level δ (so that $|\hat{\rho} - \rho| \le \hat{\varepsilon}$ holds with probability at least $1 - \delta$), using only the primitive trace sets $\{\tau_i^s\}_{i=1}^N \sim \mathcal{D}_s$ for $s \in \mathcal{S}$.

Why the problem is hard. The restriction to primitive traces is what makes the problem both useful and technically nontrivial. It is the source of the speedup over monolithic SMC: primitives are short, can be simulated in parallel, and need not be re-run when c changes. The challenge of this restriction is twofold. First, since φ may be non-Markovian, $\rho(c, \varphi)$ depends on the full trace rather than its terminal state, which rules out the Markovian compositional estimator of Yalcinkaya et al. (10). Second, in $c_1 \,;\, c_2$ the state distribution at the end of c_1 generally differs from the initial-state distribution of c_2 when c_2 is simulated as a primitive, so naively concatenating primitive traces yields a biased estimate of $\rho(c_1 \,;\, c_2, \varphi)$ unless this mismatch is explicitly corrected.

Related work. The closest prior work is Yalcinkaya et al. (10) who introduce compositional simulation-based analysis for AI-based autonomous systems but restrict their estimator to Markovian specifications; we lift this restriction by augmenting the simulation state with a DFA monitor and propagating its state distribution across primitive boundaries. Automata-based runtime verification (RV) of non-Markovian temporal properties is itself well-studied, from Peled and Havelund's first-order temporal monitors (7) to the broader RV literature, but these techniques are applied to single traces rather than composed into a satisfaction-probability estimator over composite scenarios. On the scenario-specification side, our frontend consumes Scenic (3; 9; 2) programs directly, in contrast to the custom composite format of (10), which lets us inherit Scenic's simulator-agnostic backend support and its native `do`, `do choose`, and `do shuffle` composition operators.

3 End-to-End Compositional Analysis for Safety Specifications

Given a composite Scenic program c and a non-Markovian specification $\varphi = (\mathcal{A}_\varphi, L)$, our framework returns $(\hat{\rho}(c, \varphi), \hat{\varepsilon})$ in four stages (Fig. 1): (i) recursively parse the Scenic source into a composition graph over primitives $\mathcal{S}$; (ii) simulate N traces per primitive in parallel; (iii) walk the graph, propagating a distribution over

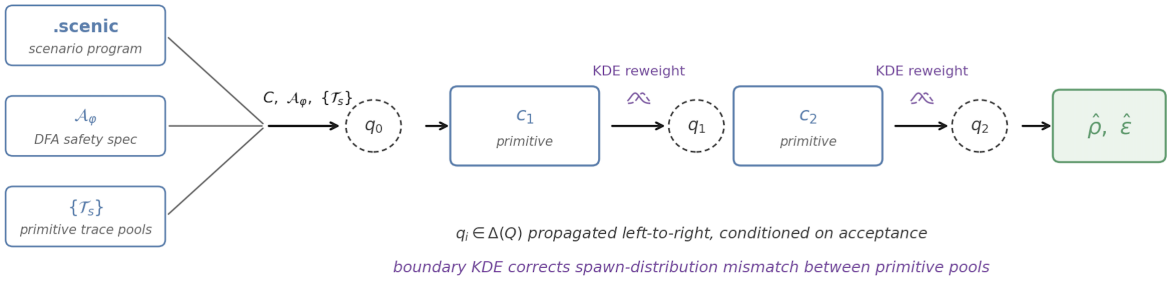

Fig. 1. End-to-end pipeline.

DFA states and correcting trace-distribution mismatch with kernel-density importance sampling; (iv) aggregate sibling branches by operator (product for sequential, weighted mixture for choose, permutation average for shuffle). Three inputs (left) feed the framework: the Scenic source `.scenic`, the DFA safety spec $\mathcal{A}_\varphi$, and per-primitive trace pools $\{\mathcal{T}_s\}$. The frontend parses the source into a linearized composition C (shown here as (c_1, c_2)); the backend propagates the DFA-state distribution $q_0 \to q_1 \to q_2$ left-to-right conditioned on acceptance, with a KDE reweight at each boundary, producing $(\hat{\rho}, \hat{\varepsilon})$.

3.1 Scenic Frontend

Scenic's composition vocabulary (do, do choose, do shuffle, and nested scenario blocks) maps directly onto the sequential, probabilistic-choice, and shuffle operators of §2, so a Scenic program already carries the compositional decomposition the analysis needs; no hand-specification of primitives is required. Working in Scenic also inherits its simulator-agnosticism (CARLA, Webots, MetaDrive, LGSVL), keeping the backend in §3.2 independent of which simulator generates the per-primitive traces.

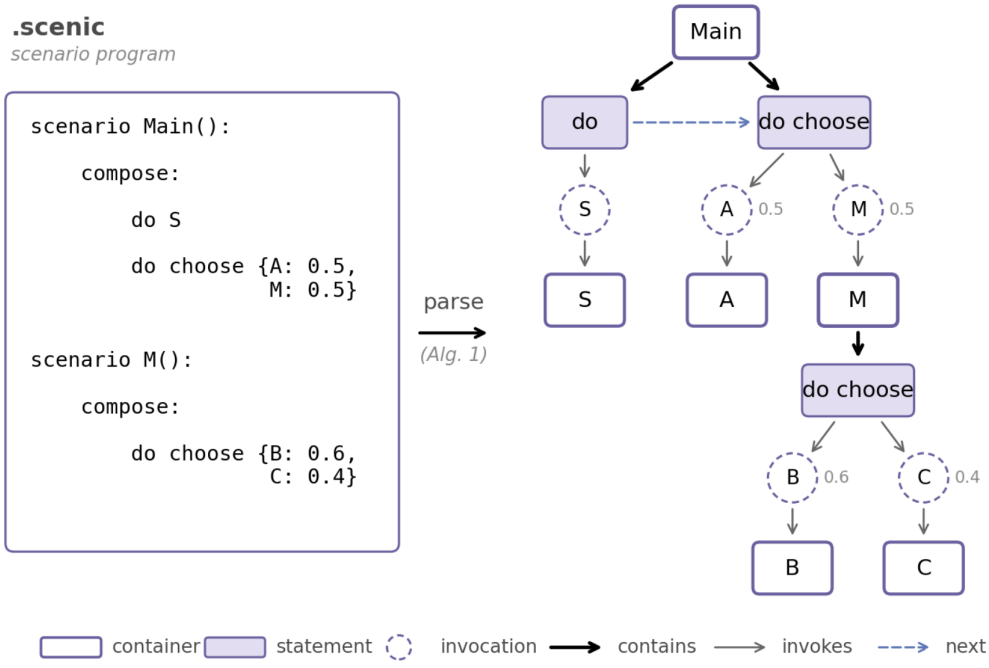

Fig. 2. Scenic source with nested choose (left) and its extracted composition DAG (right). All three node kinds (V_C containers, V_S statements, V_I invocations) and all three edge kinds (contains, invokes, next) are exercised.

Challenge. Composition in Scenic nests arbitrarily: a compose block can invoke another scenario whose body has further choose or shuffle clauses, and nested choose weights combine multiplicatively along invocation paths. A flat parse loses this structure. For example, choose$\{A{:}0.5,\ M{:}0.5\}$ with $M =$ choose$\{B{:}0.6,\ C{:}0.4\}$ is the three-way distribution (A:0.5, B:0.3, C:0.2), not the two-way distribution (A:0.5, M:0.5) obtained by reading the outer choose in isolation. Recovering $\mathcal{D}_c$ requires a recursive traversal of the Scenic abstract syntax tree (AST) that preserves operator semantics and propagates weights across invocation boundaries.

Composition graph. We represent the composite scenario as a directed acyclic graph (DAG) $G = (V, E)$ with $V = V_C \cup V_S \cup V_I$ partitioned into three node kinds. V_C contains one *container* node per scenario or behavior definition. V_S contains one *statement* node per do, do choose, or do shuffle clause, each carrying its operator label. V_I contains one *invocation* node per call site, each carrying a target identifier and (under choose) a positive weight. Edges fall into three kinds: *contains* (a container to each of its statements), *invokes* (a statement to its invocations and each invocation to its target), and *next* (between sibling statements inside a container, encoding sequential order). Fig. 2 shows the graph for a small example.

Linearization. Algorithm 1 reduces G to a list of CompositionStep values, where

$$\mathsf{CompositionStep} \;=\; \mathcal{S} \;\cup\; \Big\{\, f : \mathcal{S} \to [0,1] \;\Big|\; \textstyle\sum_{s \in \mathcal{S}} f(s) = 1 \,\Big\}.$$

Sequential statements concatenate their children's step lists; choose produces a single step that flattens the child step lists into a distribution by multiplying weights along the invocation path; shuffle over k children yields the $k!$ permutation step lists with uniform weight $1/k!$.

Algorithm 1 ParseAndLinearize.

Require: Scenic source P, entry scenario c
build DAG G from ParseScenic(P); **return** Linearize(G, c)
function Linearize(G, v)
  if v invokes primitive s **then**
    return $[\,s\,]$
  else if v is sequential over $u_1, \ldots, u_k$ **then**
    return Linearize(G, u_1); $\cdots$; Linearize(G, u_k)
  else if v is choose$\{(u_i, w_i)\}$ **then**
    return $[\,\{\, s : w_i p \mid (s, p) \in$ Flatten(Linearize(G, u_i))$\,\}\,]$
  else if v is shuffle$\{u_1, \ldots, u_k\}$ **then**
    return one step list per $\pi \in S_k$, weighted $1/k!$
  end if
end function

3.2 VerifAI Backend

Setup. The backend extends VerifAI's compositional analysis engine (1) from the Markovian estimator of (10) to DFA safety specifications. We introduce a new `automaton_specification` class in VerifAI that wraps the `dfa` library (8) to represent a safety property as a DFA $\mathcal{A}_\varphi = (Q, \Sigma, \delta, q_0, F)$ with labeling $L : X \to \Sigma$ (Figure 3), and extend the engine to consume it. It consumes a linearized composition $C = (c_1, \ldots, c_N)$ from the frontend, where each c_i is either a primitive name (sequential step) or a weighted dictionary $\{s^{(j)} : p_j\}_j$ (random choice). The backend also consumes per-primitive trace pools $\mathcal{T}_s$ and entry/exit features $\phi_{\text{src}}, \phi_{\text{tgt}}$ (e.g., position, speed). It returns $(\hat{\rho}, \hat{\varepsilon})$ estimating $\Pr[\hat{\delta}(q_0, L(\tau)) \in F]$ over the composite distribution. Algorithm 2 states the full procedure; the three challenges below describe its components, with line references back into the algorithm.

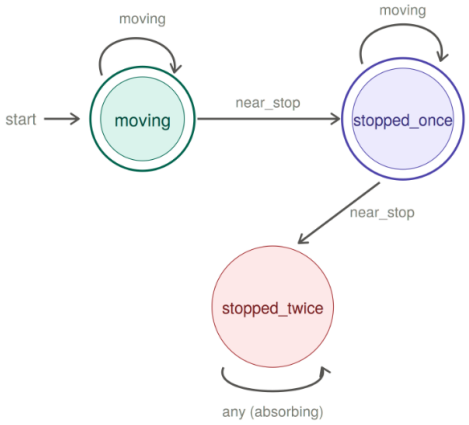

Fig. 3. Example DFA for the safety spec "No more than one near-stop." `stopped_twice` is absorbing rejecting.

Algorithm 2 CheckWithDFA: compositional SMC for DFA safety specs.

Require: Composition $C = (c_1, \ldots, c_N)$; spec $(\mathcal{A}_\varphi, L)$; pools $\{\mathcal{T}_s\}$; features $\phi_{\text{src}}, \phi_{\text{tgt}}$; confidence δ
Ensure: $(\hat{\rho}, \hat{\varepsilon})$
1: $\delta_i \leftarrow \delta/N$ *// Bonferroni per-step budget*
// Forward pass: precompute pre-step DFA distributions
2: $q_{\text{dist}}^{(1)} \leftarrow \mathbf{1}_{q_0}$
3: **for** $i = 1, \ldots, N-1$ **do**
4: $q_{\text{dist}}^{(i+1)} \leftarrow$ accept-conditioned mixture successor of $q_{\text{dist}}^{(i)}$ under c_i
5: **end for**
// Main estimation loop
6: $p_{\text{src}} \leftarrow \emptyset$
7: **for** $i = 1, \ldots, N$ **do**
8: Parse c_i as $\{(s^{(j)}, p_j)\}_j$ ($p_j = 1$ if sequential)
9: **for** each branch j **do**
10: $\ell_i(\tau) \leftarrow \sum_q q_{\text{dist}}^{(i)}(q)\, \mathbf{1}[\hat{\delta}(q, L(\tau)) \in F]$ for $\tau \in \mathcal{T}_{s^{(j)}}$
11: **if** $i = 1$ **then**
12: $\tilde{w}_i(\tau) \leftarrow 1$ *// uniform weights, no handoff yet*
13: **else**
14: $p_{\text{tgt}}^{(j)} \leftarrow$ KDE over $\phi_{\text{src}}(\mathcal{T}_{s^{(j)}})$
15: $\tilde{w}_i(\tau) \leftarrow p_{\text{src}}(\phi_{\text{src}}(\tau))/p_{\text{tgt}}^{(j)}(\phi_{\text{src}}(\tau))$
16: **end if**
17: $\hat{\rho}_{i,j} \leftarrow \sum_\tau \tilde{w}_i(\tau)\, \ell_i(\tau) \big/ \sum_\tau \tilde{w}_i(\tau)$ *// self-normalized IS estimate*
18: $N_{\text{eff}} \leftarrow \big(\sum_\tau \tilde{w}_i(\tau)\big)^2 \big/ \sum_\tau \tilde{w}_i(\tau)^2$
19: $\hat{\varepsilon}_{i,j} \leftarrow \sqrt{\log(2/\delta_i)/(2N_{\text{eff}})}$ *// Hoeffding bound with effective sample size*
20: **end for**
21: $\hat{\rho}_i \leftarrow \sum_j p_j\, \hat{\rho}_{i,j}$; $\hat{\varepsilon}_i^{\,2} \leftarrow \sum_j p_j^2\, \hat{\varepsilon}_{i,j}^2$
22: **if** $\hat{\rho}_i = 0$ **then return** $(0, 0)$
23: **end if** *// early termination*
24: $p_{\text{src}} \leftarrow$ weighted KDE over ϕ_{tgt} of accepting traces, pooled across branches with weights $\propto p_j$
25: **end for**
26: **return** $\hat{\rho} = \prod_i \hat{\rho}_i$, $\hat{\varepsilon} = \hat{\rho}\sqrt{\sum_i (\hat{\varepsilon}_i/\hat{\rho}_i)^2}$

Challenge 1: non-Markovian satisfaction. The Markovian estimator labels each trace by $\mathbf{1}[L(x_T)]$; DFA satisfaction instead depends on the full labeled sequence and on the entry state. We maintain a distribution $q_{\text{dist}}^{(i)} \in \Delta(Q)$ over DFA entry states, conditioned across primitives on acceptance, and label each trace by

$$\ell_i(\tau) = \sum_{q \in Q} q_{\text{dist}}^{(i)}(q)\, \mathbf{1}\big[\hat{\delta}(q, L(\tau)) \in F\big]$$

(line 10 of Algorithm 2). The accept-conditioning is essential: advancing $q_{\text{dist}}^{(i+1)}$ over *all* step-i traces (including ones already in absorbing rejecting states) deflates both $\hat{\rho}_i$ and $\hat{\rho}_{i+1}$, so $\prod_i \hat{\rho}_i$ underestimates the truth. We therefore compute the full sequence $\{q_{\text{dist}}^{(i)}\}_{i=1}^N$ in a *forward pass* over C (lines 1–5) before the main estimation loop, so each step uses the correct pre-step distribution.

Challenge 2: handoff distribution mismatch. Each primitive is simulated standalone, so its entry distribution over X differs from its predecessor's exit distribution; naive concatenation is biased. We correct with self-normalized importance sampling (line 15): fit a Gaussian KDE $p_{\text{src}}^{(i)}$ to step $i{-}1$'s accepting-exit features ϕ_{tgt} and $p_{\text{tgt}}^{(i)}$ to step i's entry features ϕ_{src}, with

$$\tilde{w}_i(\tau) = \frac{p_{\text{src}}^{(i)}(\phi_{\text{src}}(\tau))}{p_{\text{tgt}}^{(i)}(\phi_{\text{src}}(\tau))}, \qquad w_i(\tau) = \tilde{w}_i(\tau) \big/ \textstyle\sum_{\tau'} \tilde{w}_i(\tau').$$

The per-step estimate is the self-normalized IS-weighted mean $\hat{\rho}_i = \sum_\tau \tilde{w}_i(\tau)\, \ell_i(\tau) / \sum_\tau \tilde{w}_i(\tau)$ (line 17). Near-disjoint supports collapse the weights, which we mitigate by *prewarming* primitives so successive supports overlap, and by monitoring effective sample size.

Challenge 3: random-choice composition. A step $c_i = \{s^{(j)} : p_j\}_j$ branches all three machinery components:

$$\hat{\rho}_i = \sum_j p_j \, \hat{\rho}_{i,j}, \qquad q_{\text{dist}}^{(i+1)} = \sum_j p_j \, \tilde{q}_{\text{dist}}^{(i+1),j},$$

where $\hat{\rho}_{i,j}$ is the per-branch IS estimate from a shared $q_{\text{dist}}^{(i)}$ and $\tilde{q}_{\text{dist}}^{(i+1),j}$ is its accept-conditioned successor (line 21). When step $i{+}1$ fits its source KDE, accepting-exit features are pooled across branches with weights $\propto p_j$ via `scipy.stats.gaussian_kde`'s `weights` argument (line 24), so the mixture mass is faithful rather than implicitly uniform. Sequential steps are the degenerate $|j|{=}1$, $p_j{=}1$ case.

Aggregation. The chain rule

$$\Pr\Big[\bigwedge_{i=1}^{N} \text{accept}_i\Big] = \prod_{i=1}^{N} \Pr[\text{accept}_i \mid \text{accept}_{<i}]$$

justifies multiplying per-step estimates into the composite estimate, with per-step Hoeffding bounds $\hat{\varepsilon}_i$ (using the effective sample size N_{eff} of the importance weights, line 19) propagated as

$$\hat{\rho} = \prod_{i=1}^{N} \hat{\rho}_i, \qquad \hat{\varepsilon} = \hat{\rho} \sqrt{\textstyle\sum_{i=1}^{N} (\hat{\varepsilon}_i / \hat{\rho}_i)^2}, \qquad \hat{\varepsilon}_i^2 = \textstyle\sum_j p_j^2 \, \hat{\varepsilon}_{i,j}^2.$$

A union bound over the N per-step events justifies the $\delta_i = \delta/N$ split (line 1) so that the composite estimate holds at overall confidence $1 - \delta$.

Summary of innovations. Algorithms 3 and 4 place the Markovian baseline of (10) side-by-side with our DFA-augmented version, with all changes highlighted in blue. The three additions correspond directly to Challenge 1: line 1 of Algorithm 4 initializes the DFA-state distribution, line 3 replaces the trace's Boolean label with its acceptance probability $\ell_i(\tau) = \Pr_{q \sim q_{\text{dist}}}[\hat{\delta}(q, L(\tau)) \in F]$, and line 6 conditions q_{dist} on acceptance before the next iteration. The KDE handoff machinery and the per-step estimate (lines 4–5) are unchanged from the baseline; non-Markovian satisfaction therefore enters in exactly three lines, while Challenges 2 and 3 fold into existing structure without altering the algorithmic skeleton.

Algorithm 3 Before: Markovian.

1: **for** each scenario s_i **do**
2: $\quad \ell(\tau) \leftarrow \mathbf{1}[\tau.\text{label}]$
3: $\quad$ Build KDEs, compute IS weights $w(\tau)$
4: $\quad \hat{\rho}_i \leftarrow \sum_\tau w(\tau)\, \ell(\tau)$
5: $\quad p_{\text{src}} \leftarrow$ empirical over accepted τ
6: **end for**
7: **return** $\hat{\rho} = \prod_i \hat{\rho}_i$

Algorithm 4 After: with DFA.

1: $q_{\text{dist}} \leftarrow \mathbf{1}_{q_0}$
2: **for** each scenario s_i **do**
3: $\quad \ell(\tau) \leftarrow \Pr_{q \sim q_{\text{dist}}}[\hat{\delta}(q, L(\tau)) \in F]$
4: $\quad$ Build KDEs, compute IS weights $w(\tau)$
5: $\quad \hat{\rho}_i \leftarrow \sum_\tau w(\tau)\, \ell(\tau)$
6: $\quad q_{\text{dist}} \leftarrow q_{\text{dist}} \mid \text{accept}$
7: $\quad p_{\text{src}} \leftarrow$ empirical over accepted τ
8: **end for**
9: **return** $\hat{\rho} = \prod_i \hat{\rho}_i$

Scope. A do choose whose branches themselves expand into multi-step sequences cannot be collapsed into a single distributional step by Algorithm 1; the frontend instead enumerates the root-to-leaf paths π with their priors p_π and the backend evaluates each path independently, returning $\hat{\rho} = \sum_\pi p_\pi \, \hat{\rho}(\pi)$. Parallel composition (`do A(), B()` at the same simulation time) is out of scope: there is no single physical handoff boundary at which to apply the KDE correction. Non-safety specifications are deferred to §5.

4 Experiments

We ask two questions. **Q1 (accuracy)**: at a matched *trace* budget, does compositional analysis recover the monolithic ground truth? **Q2 (efficiency)**: at a matched *wall-clock* budget, does it deliver tighter confidence intervals? The one-line answers are *yes* and *yes*: compositional estimates agree with the monolithic baseline within Hoeffding error on every (c, φ) pair, and reach the same precision in 4–7× less wall-clock time.

4.1 Setup

Composite scenarios. We create three sets of scenarios for evaluation. *Set A* consists of five sequential composites {SX, SXS, SC, SCS, SCX} over the primitives Straight, Intersection, and Curve. *Set B* is the five-step wander composite

$$\mathsf{Main} \;=\; 5 \times \mathsf{choose}\{\text{GoStraight, TurnLeft, TurnRight, Brake}\},$$

yielding $4^5 = 1024$ distinct execution paths over four leaf primitives; N=5: this is the composite length, not the number of evaluated paths. It carries both the accuracy result of §4.2 (Eq. (1)) and the multi-spec decoupling demonstration of Appendix A.2. *Set C* is the two-step approach-then-turn composite $\mathrm{Sub}_1 \,;\, \mathsf{choose}\{\mathrm{Sub}_2^L, \mathrm{Sub}_2^R, \mathrm{Sub}_2^S\}$, used in §4.3 to isolate the wall-clock comparison under the Markovian `safe_under_max` spec; Markovianity of the monitor does not affect the cost comparison being asked there.

Simulator. All simulation runs in MetaDrive (6) in two-dimensional mode at a 0.1 s timestep, on top of VerifAI (1). Set A drives MetaDrive directly via its map-string backend; Set B exercises the full Scenic (4) → MetaDrive pipeline contributed in this paper, with Scenic specifying composite structure and per-primitive worlds.

Specifications. We evaluate three safety properties, each a DFA with an absorbing rejecting state. φ_{toll} rejects after the third near-stop; φ_{stop} tightens this to a single near-stop; φ_{brake} rejects after the second hard brake. Formal DFA constructions, labeling thresholds, and trace counts are deferred to Appendix A.

Baseline. The monolithic baseline simulates each composite as a single chained Scenic scenario; $\hat{\rho}_{\text{mono}}$ is the empirical DFA accept rate. Both methods report Hoeffding half-widths at $\delta = 0.05$.

4.2 Q1: Accuracy at matched trace budget

Given a matched trace budget M (M traces per primitive on the compositional side and M composite traces on the monolithic side), the two estimators agree within their Hoeffding intervals for every (c, φ) pair we test. Table 1 reports Set A results under both φ_{toll} and φ_{stop}, and Eq. (1) reports the Set B result under φ_{brake}.

Table 1. Set A on MetaDrive, $\hat{\rho} \pm \hat{\varepsilon}$ at δ=0.05. Every $|\Delta\hat{\rho}|$ falls inside $\hat{\varepsilon}_{\text{mono}} + \hat{\varepsilon}_{\text{comp}}$, i.e., the two estimators agree within statistical error on every (c, φ) pair.

	φ_{toll} (k=3 near-stops)		φ_{stop} (at most one near-stop)	
Composite	$\hat{\rho}_{\text{mono}}$	$\hat{\rho}_{\text{comp}}$	$\hat{\rho}_{\text{mono}}$	$\hat{\rho}_{\text{comp}}$
SX	0.702 ± 0.061	0.727 ± 0.092	0.650 ± 0.061	0.692 ± 0.097
SXS	0.664 ± 0.061	0.713 ± 0.108	0.600 ± 0.061	0.687 ± 0.114
SC	0.860 ± 0.061	0.895 ± 0.098	0.874 ± 0.061	0.904 ± 0.105
SCS	0.862 ± 0.061	0.892 ± 0.123	0.866 ± 0.061	0.897 ± 0.129
SCX	0.620 ± 0.061	0.652 ± 0.101	0.584 ± 0.061	0.633 ± 0.104

For Set B at N=5 under φ_{brake}:

$$\hat{\rho}_{\text{mono}} = 0.896 \pm 0.043, \qquad \hat{\rho}_{\text{comp}} = 0.868 \pm 0.051. \tag{1}$$

The same Set B primitive trace pool is additionally evaluated against seven further DFAs without re-running MetaDrive, illustrating that compositional analysis decouples specifications from simulation; see Appendix A.2.

Interpretation. Compositional point estimates land slightly above the monolithic baseline on Set A across all five composites, but the gap is within Hoeffding error in every row and shows no trend with composite length. This is evidence that the accept-conditioning fix in `check_with_dfa` (§3.2) is removing the dominant source of compositional drift, since a residual multiplicative bias from un-conditioned rejection mass would grow with the number of primitives. The most informative single row is SCX under φ_{stop}: this is where rejection mass is largest (intersections induce near-stops) and where the conditioning fix has the most leverage. On Set B the compositional interval is the wider of the two, reflecting the structural cost of multiplicative variance accumulation along an N=5 chain.

4.3 Q2: Efficiency at matched wall-clock budget

Protocol. For each method we run one continuous simulation per composite up to $T_{\max}$ and record per-primitive trace counts at periodic checkpoints. The estimator is evaluated on the trace prefix available at each checkpoint, yielding a sequence $(T, \hat{\rho}, \hat{\varepsilon})$ from a single run per method. The configuration used to produce Fig. 4 is detailed in App. A.3.

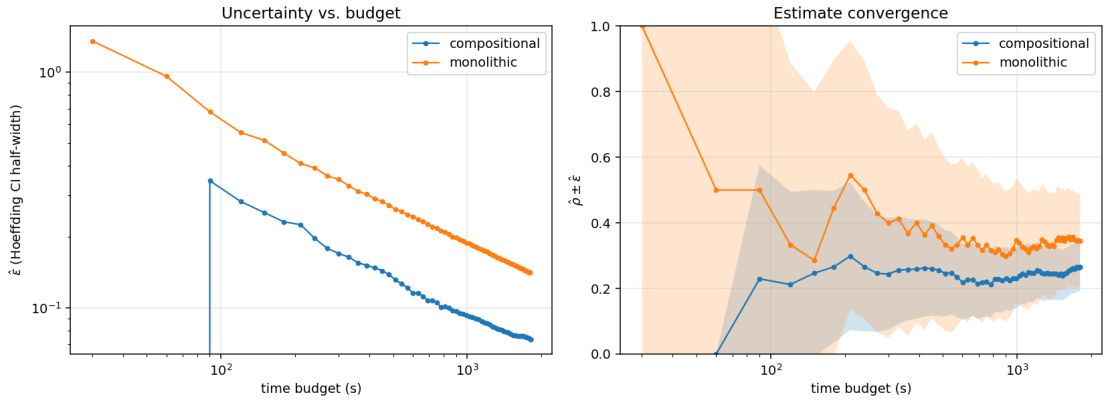

Fig. 4. Set C (§4.1) under the `safe_under_max` DFA (App. A.3). **Left:** $\hat{\varepsilon}$ as a function of wall-clock budget T on log-log axes. Past the simulator startup window the compositional curve sits strictly below the monolithic curve at every checkpoint, with a log-log offset that ranges from 4× at $\hat{\varepsilon}^* = 0.20$ to 7× at $\hat{\varepsilon}^* = 0.15$ over the measured range. **Right:** $\hat{\rho}$ with shaded $\pm\hat{\varepsilon}$ band as a function of T. The two methods agree to within their confidence intervals throughout, so the precision advantage in the left panel is not bought through bias. The compositional curve only begins once all four parallel primitive workers have finished MetaDrive startup and emitted their first trace.

Result. Past a brief simulator-startup window, the compositional confidence interval $\hat{\varepsilon}(T)$ sits strictly below the monolithic one at every checkpoint (Fig. 4, left). The two point estimates $\hat{\rho}(T)$ agree to within their confidence intervals throughout the sweep (Fig. 4, right), so the precision advantage is not bought through bias. Reading the wall-clock ratio $T_{\text{mono}}/T_{\text{comp}}$ at fixed target precision off the left panel yields $\approx 4\times$ at $\hat{\varepsilon}^* = 0.20$ and $\approx 7\times$ at $\hat{\varepsilon}^* = 0.15$ over the measured range.

Source of the gap. Two structural advantages drive the result on Set C. First, the compositional pipeline spawns one parallel worker per primitive (Sub_1 together with the three turning branches), four workers in total, while the monolithic pipeline runs a single worker on the chained behavior. Second, each primitive trace covers only one composition step and is therefore a factor of two shorter than a chained composite trace under matched horizons. A third structural advantage, trace-pool reuse across composites sharing the same primitives, is not exercised in this single-composite sweep but would compound the gap on a corpus of composites built from overlapping primitive sets.

5 Discussion & Future work

Summary of results. We extended VerifAI's compositional analysis from Markovian to non-Markovian safety properties via a left-to-right DFA-state distribution that is conditioned on acceptance at every handoff, keeping $\hat{\rho}_{\text{comp}}$ from drifting low as composite length grows.

On the MetaDrive Set A primitives, the compositional estimator agrees with the monolithic baseline within Hoeffding error on every (c, φ) pair under both φ_{toll} and φ_{stop} (Table 1).

On the Scenic-graph Set B chain at N=5 under φ_{brake}, the same agreement holds: $|\Delta\hat{\rho}| = 0.028$, well inside both Hoeffding intervals (Eq. (1)).

On Set C, the compositional confidence interval $\hat{\varepsilon}(T)$ sits 4–7× below the monolithic curve at matched precision across the measured range (Fig. 4).

The same Set B trace pool further supports a battery of seven additional DFAs at $\sim$ 1 s of post-hoc Python each, illustrating that compositional analysis decouples specifications from simulation.

What didn't work, and why. Two failure modes shaped the final design.

First, naive sequential composition without rejection-conditioning produces a multiplicative bias that grows with composite length: rejection mass is counted both by the per-step accept rate and by the propagated DFA distribution, and the estimator drifts low on longer composites.

Conditioning the forward DFA-distribution on acceptance at each handoff (Algorithm 4) fixes this, and the residual gap between $\hat{\rho}_{\text{comp}}$ and $\hat{\rho}_{\text{mono}}$ in our final estimates shows no trend with composite length, evidence that the fix is doing what it should.

Second, the KDE-weight collapse flagged in §3.2 (Challenge 2) showed up empirically on both backends; we addressed it with per-primitive entry-state calibration (spawn-distance and initial-speed tuning on Set A, randomized cruise prewarm on Set B; see Appendix A). The underlying sensitivity remains a real cost: adding a new primitive requires repeating this calibration.

Future work: bounded-liveness specifications. This paper restricts attention to safety specifications, DFAs with an absorbing rejecting state where, once acceptance is lost, it cannot be recovered.

A natural extension is bounded-liveness via complementation: on finite traces of length T, a bounded-liveness property such as "the vehicle must exceed $v_{\max}$ at some point" equals $1 - \rho(\neg\varphi)$, where $\neg\varphi$ is the flat safety property "the vehicle never exceeds $v_{\max}$ within the trace."

Where the negation is itself a flat safety DFA, this trick is sound and works with our existing estimator without modification; the machinery transfers directly, and we view this as a near-term extension rather than a separate piece of theory.

The trick does *not* generalize beyond the bounded-reachability fragment: bounded-response, bounded-sequenced-reachability, bounded persistence, and bounded recurrence all negate to formulas with nested temporal operators rather than flat safety DFAs, and so cannot be reduced to a single absorbing-reject monitor.

Compositional treatment of those classes requires either a product construction across multiple "live" DFA regions or, in the unbounded case, reasoning about infinite-suffix acceptance, and is left to future work.

A near-term path that stays within the present machinery is bounded response ("within k steps of A, observe B"), which is expressible as a finite-window safety DFA and captures a large fraction of practical AV requirements without leaving the absorbing-reject framework.

Future work: additional simulator backends. The Scenic frontend is simulator-agnostic by construction, and our parser already targets CARLA, Webots, etc. in addition to MetaDrive. We were not, however, able to include CARLA results in the present evaluation. The shared compute cluster we had access to ran a CARLA version older than the one our pipeline targets, and upgrading it in place was not an option since other users on the cluster depended on that exact version. The natural workaround, running our own CARLA build inside a Docker container on the same machine, was feasible in principle but ruled out by the project timeline, and the resulting delay propagated to the other simulator backends as well. Closing this gap is straightforward engineering rather than a research question: a containerized CARLA build, paired with the existing Scenic $\rightarrow$ CARLA adapter, would let us replicate the Set B chain study under CARLA's richer traffic and sensor models without any change to the backend described in §3.2.

Future work: other directions. Two further directions follow naturally from the present framework.

(i) Official VerifAI integration. The current implementation lives as an extension of CompositionalAnalysisEngine; upstreaming the DFA monitor and Scenic parser into VerifAI proper would expose them to the existing falsifier pipeline.

(ii) Adaptive primitive calibration. The boundary-distribution overlap that compositional estimation requires is currently tuned by hand (§A); a diagnostic based on per-handoff effective sample size, paired with an automatic spawn/prewarm adjustment, would remove the manual calibration step and make the framework usable on new primitive libraries without per-scenario tuning.

Roles on the Team

Abhi led the compositional analysis backend: the DFA-augmented state-distribution propagation, the rejection-conditioning fix in `check_with_dfa`, and the experimental evaluation on MetaDrive.

Arya led the Scenic frontend, including the recursive DAG parser over containers, composition statements, and weighted invocations, and the linearization that feeds the backend.

Abhi also handled the end-to-end integration of the frontend and backend into a single Scenic → MetaDrive pipeline.

Class Topics and Feedback

This project drew most directly on three blocks of 219C material. Modeling for verification and temporal-logic specification (3/2, 3/4) gave us the vocabulary for non-Markovian safety properties as finite-state monitors; the runtime verification and assurance lecture (4/27) connected DFA monitors to traces from a running system rather than a static model; and the lecture on verification of AI/ML systems via simulation and probabilistic verification (4/22) framed the wall-clock-budgeted statistical model checking question we ended up answering.

One topic we wished had landed slightly earlier in the semester was compositional reasoning (4/1). The compositional decomposition is the core mechanism of our pipeline, and having that material earlier in the term would have shortened the time we spent rediscovering why naive product estimators fail when consecutive primitives have non-overlapping handoff distributions.

One topic we would have benefited from is a deeper treatment of *importance sampling and its variance properties* in the context of probabilistic verification. Our self-normalized IS estimator and the KDE re-weighting at handoff points were among the hardest parts of the project to get right, and a lecture-level treatment of ESS-based diagnostics would have saved several debugging cycles.

Acknowledgements

We thank Beyazit Yalcinkaya and Kai-Chun Chang for active advising throughout the project, including discussions on the compositional analysis framework, feedback on the DFA monitor design, and guidance on the Scenic frontend. We also thank Sanjit Seshia for research direction and feedback on the project's scope and positioning. This work was carried out as a course project for EECS 219C: Formal Methods: Specification, Verification, and Synthesis at UC Berkeley, Spring 2026.

Bibliography

[1] Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Ravanbakhsh, H., Vazquez-Chanlatte, M., Seshia, S.A.: VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In: Computer Aided Verification (CAV 2019). LNCS, vol. 11561, pp. 432–442. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_25

[2] Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: A language for scenario specification and scene generation. In: Programming Language Design and Implementation (PLDI) (2019)

[3] Fremont, D.J., Kim, E., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: A language for scenario specification and data generation. Machine Learning (2022)

[4] Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., Mehta, S.: Scenic: A language for scenario specification and data generation (2021)

[5] Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., et al. (eds.) Runtime Verification (RV 2010). LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16612-9_11

[6] Li, Q., Peng, Z., Hong, L., Zhou, B.: MetaDrive: Composing diverse driving scenarios for generalizable reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2021)

[7] Peled, D., Havelund, K.: Efficient runtime verification of first-order temporal properties. Tech. rep., Jet Propulsion Laboratory, NASA, Pasadena, CA (2018), https://ntrs.nasa.gov/citations/20210008459, nASA Technical Reports Server, Document ID 20210008459

[8] Vazquez-Chanlatte, M.: dfa: A simple Python implementation of a DFA. https://github.com/mvcisback/dfa (2024), python package on PyPI; MIT licensed

[9] Vin, E., Kashiwa, S., Rhea, C., Fremont, D.J., Kim, E., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: 3D environment modeling for falsification and beyond with Scenic 3.0. In: Computer Aided Verification (CAV) (2023)

[10] Yalcinkaya, B., Torfah, H., Fremont, D.J., Seshia, S.A.: Compositional simulation-based analysis of AI-based autonomous systems for Markovian specifications. In: Katsaros, P., Nenzi, L. (eds.) Runtime Verification (RV 2023). LNCS, vol. 14245, pp. 191–212. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-44267-4_10

A Experimental Details

A.1 Set A: MetaDrive primitives

Harness. Set A is driven directly through MetaDrive's map-string backend. Primitives {S, X, C} (Straight, Intersection, Curve) and the five monolithic ground-truth composites {SX, SXS, SC, SCS, SCX} are each instantiated with a fixed PRNG seed (0–2 for primitives, 3–7 for monolithics) and rolled out for 500 MetaDrive episodes under the expert policy. To induce the near-stop events that the Set A DFAs respond to, each episode (i) samples an initial ego speed uniformly from 40–90 km/h (Uniform$(40/3.6, 90/3.6)$ m/s) and assigns it along the current-lane direction, and (ii) with probability 0.6, spawns randomly placed obstacles ahead of the ego that force braking. The obstacle and initial-speed RNGs are seeded as seed $+ ep$ on episode ep so the per-scenario seeds above fully determine each run. Trace CSVs record $(x, y, \text{heading}, \text{speed}, \text{action}, \text{reward})$ per step. The compositional estimator consumes only the three primitive CSVs; the monolithic estimator consumes a single full-composite CSV. KDE features are (x, y, speed) with the first two centered at the previous step's accepting handoff.

Labeling. Both Set A DFAs share a single per-row labeling function that emits a *near-stop* event when speed < 3.5 m/s and *moving* otherwise. The two DFAs differ only in how they *count* these events.

Prewarming for handoff support. Compositional estimation collapses when consecutive primitives have non-overlapping spawn/exit distributions: the KDE importance weights at the handoff become ≈ 0, and $\hat{\rho}_{\text{comp}}$ collapses to the leading primitive's per-step value (§3.2). For each MetaDrive primitive we therefore (i) tune the spawn distance to the upcoming feature, e.g., distance to the next intersection in X, or to the start of the curve in C, and (ii) cap the obstacle-spawning region ahead of the ego so that the expert policy's terminal speed at the end of primitive i falls within the support of primitive $i{+}1$'s entry speed. The initial speed range Uniform$(40/3.6, 90/3.6)$ m/s reported above is itself part of this calibration: it widens the entry-speed support of every primitive so a handoff from any predecessor has a non-zero KDE response. We monitored the effective sample size (ESS) at each handoff as a diagnostic; configurations with ESS $< 0.1N$ on any handoff (where N is the per-primitive trace count) were re-tuned before running the comparison against monolithic SMC.

φ_{toll} *(Tollgate, K=3).* States $Q = \{\mathsf{moving}, \mathsf{wait}_1, \mathsf{wait}_2, \mathsf{wait}_3, \mathsf{violated}\}$; start moving; accepting set $F = Q \setminus \{\mathsf{violated}\}$. The intuition is "once you slow down, you must stay slow for at least three consecutive steps before resuming." Transitions:

$$\begin{aligned} \mathsf{moving}, \text{slow} &\mapsto \mathsf{wait}_1 & \mathsf{moving}, \text{fast} &\mapsto \mathsf{moving} \\ \mathsf{wait}_i, \text{slow} &\mapsto \mathsf{wait}_{i+1} \ \ (i < 3) & \mathsf{wait}_i, \text{fast} &\mapsto \mathsf{violated} \ \ (i < 3) \\ \mathsf{wait}_3, \text{slow} &\mapsto \mathsf{moving} & \mathsf{wait}_3, \text{fast} &\mapsto \mathsf{violated} \\ \mathsf{violated}, * &\mapsto \mathsf{violated}. \end{aligned}$$

φ_{stop} *(at most one near-stop).* States $Q = \{\mathsf{moving}, \mathsf{stopped\_once}, \mathsf{stopped\_twice}\}$; start moving; accepting set $F = Q \setminus \{\mathsf{stopped\_twice}\}$. Transitions:

$$\begin{aligned} \mathsf{moving}, \text{near\_stop} &\mapsto \mathsf{stopped\_once} & \mathsf{moving}, \text{moving} &\mapsto \mathsf{moving} \\ \mathsf{stopped\_once}, \text{near\_stop} &\mapsto \mathsf{stopped\_twice} & \mathsf{stopped\_once}, \text{moving} &\mapsto \mathsf{stopped\_once} \\ \mathsf{stopped\_twice}, * &\mapsto \mathsf{stopped\_twice}. \end{aligned}$$

A.2 Set B: Scenic → MetaDrive wander composition

Composition. The Scenic source `composed_wander.scenic` defines

```
Main = 5 × do choose {GoStraight: 1, TurnLeft: 1, TurnRight: 1, Brake: 1},
```

i.e., five *sequential* equal-weight choose nodes over four leaf primitives, written as five separate `do choose` statements inside one `compose` block (a single `for` loop is deliberately avoided because Scenic's parser collapses it into a single parallel-style step). This yields $4^5 = 1024$ distinct execution paths. The frontend parses Main into a linearized composition $C = (c_1, \ldots, c_5)$ where each c_i is the dictionary

$$\texttt{GoStraight}{:}\tfrac{1}{4},\ \texttt{TurnLeft}{:}\tfrac{1}{4},\ \texttt{TurnRight}{:}\tfrac{1}{4},\ \texttt{Brake}{:}\tfrac{1}{4}$$

`check_with_dfa_scenic` evaluates all 1024 paths via importance sampling over a single shared primitive trace pool; no path-by-path simulation is required. The Brake primitive in particular is what gives several of the non-Markovian specs in Table 2 a non-trivial rejection mass, e.g., `at_most_one_brake`, `k_consec_slow`, and `rise_then_fall` all require traces that actually slow down.

Leaf primitives. Each of the four primitives is a Scenic behavior that issues raw (`throttle`, `brake`, `steer`) actions every tick. GoStraight, TurnLeft, and TurnRight cruise at `throttle` $\sim$ Range$(0.3, 0.6)$ with steer 0, Range$(-0.4, -0.2)$ (negative = left), or Range$(0.2, 0.4)$ respectively. Brake applies `brake_force` $\sim$ Range$(0.5, 1.0)$ with zero throttle. Each ego is spawned along the spawn lane of a four-way intersection on Town07, offset by `DISTANCE_TO_INTERSECTION` $\sim$ Range$(-15, -5)$ m from the intersection center.

Prewarming for handoff support. Compositional estimation requires that each primitive's *entry-state* distribution overlap the previous primitive's *exit-state* distribution; otherwise the KDE importance weights at the handoff collapse to ≈ 0. Set A handles this through spawn-distance and initial-speed tuning of the MetaDrive map-string backend; Set B has no spawn-distance knob (each primitive is launched from rest at a fixed lane offset) and instead prepends a randomized *cruise prewarm* inside every leaf behavior: a Uniform$(\{0, 5, 10, 15, 25, 35\})$ tick window during which the ego applies `throttle` $\sim$ Range$(0.4, 0.8)$, zero brake, zero steer, before its actual primitive actions begin. The harness then trims the first `PREWARM_TRIM`=35 rows of each per-primitive CSV post-generation, so the recorded step 0 represents a *warm state* with varied speed across traces rather than a rest condition. This widens the entry-speed support of every primitive so that a handoff from any predecessor yields non-zero KDE weight, the same goal as the spawn-distance calibration in Set A.

Trace generation. We generate 30 MetaDrive traces per primitive (parallel workers), each `MAX_STEPS`=75 ticks long. After the `PREWARM_TRIM`=35 rows are dropped this leaves $\sim$ 40 useful ticks per primitive, matching the `WANDER_SEGMENT_LEN`=40 per-segment length used by the monolithic counterpart so the two estimators see comparable per-segment trace lengths.

Monolithic counterpart. `MonolithicWander` is the literal continuous-drive analogue of the five-step composition: a single MetaDrive simulation whose ego executes five back-to-back 40-tick segments under the `Wander` behavior. Inside `Wander`, each segment's action tuple s_k is drawn independently at scene creation from Uniform(`GO_TUPLE`, `LEFT_TUPLE`, `RIGHT_TUPLE`, `BRAKE_TUPLE`), where the four tuples are the central-tendency (`throttle`, `brake`, `steer`) values of the corresponding leaf primitives: $(0.4, 0, 0)$, $(0.4, 0, -0.3)$, $(0.4, 0, 0.3)$, and $(0, 1.0, 0)$. Per-segment action tuples are sampled independently, mirroring the per-primitive independence assumption of the compositional estimator; the monolithic run therefore measures Pr[spec satisfied | 5 independent segments], which is the same quantity the compositional estimator approximates. A run is $5 \times$ `WANDER_SEGMENT_LEN` $= 200$ ticks long; we generate 30 monolithic traces in total.

DFA specifications. Table 2 summarizes the eight DFAs defined against the shared Set B trace pool. The first three are Markovian (single-row speed-threshold) safety/liveness properties; the rest are genuinely non-Markovian, requiring DFA state beyond a per-row predicate. All eight labeling functions gate on a post-warmup window of `WARMUP_STEPS`=5 ticks. The φ_{brake} result quoted in Eq. (1) of the main text corresponds to the `at_most_one_brake` entry. The remaining seven specs are defined and exercised end-to-end by the harness against the same trace pool, but per-spec result tables are deferred to a future expanded evaluation.

Table 2. The eight DFA specifications defined against the shared Set B trace pool. "Slow" means speed $< 0.5\,\mathrm{m/s}$ and "fast" means speed $\geq 1.5\,\mathrm{m/s}$ under the per-spec labeling functions. Thresholds were tuned so each spec lands in a graded regime (rather than always accept or always reject) given the empirical post-warmup speed distribution of the wander composition.

Spec	Type	Acceptance condition (post-warmup)
`reach_high`	Markov. liveness	$\exists t:\ \mathrm{speed}_t \geq 3.0$
`safe_under_max`	Markov. safety	$\forall t:\ \mathrm{speed}_t \leq 5.5$
`reach_low_speed`	Markov. liveness	$\exists t:\ \mathrm{speed}_t \geq 1.0$
`rise_then_fall`	non-Markov.	$\exists t_1 < t_2:\ \mathrm{speed}_{t_1} \geq 3.0 \wedge \mathrm{speed}_{t_2} \leq 0.5$
`k_consec_slow`	non-Markov.	no run of > 3 steps with speed < 0.5
`k_consec_fast`	non-Markov.	no run of > 3 steps with speed ≥ 1.5
`at_most_one_brake`	non-Markov.	≤ 1 debounced slow $\rightarrow$ fast transitions
`bounded_slow_160`	non-Markov.	total slow steps across all 5 segments ≤ 160

A.3 Set C: Wall-clock sweep experiment (§4.3)

Composite. The Scenic source `composed_scenarios.scenic` defines

```
Main = Subscenario1 ; do choose {Subscenario2L: 1, Subscenario2R: 1, Subscenario2S: 1},
```

an approach primitive followed by an equal-weight three-way turning choice at a four-way intersection. The frontend parses this into a length-two linearized composition $C = (c_1, c_2)$ with $c_1 = \mathrm{Sub}_1$ and $c_2 = \{\mathrm{Sub}_2^L{:}\frac{1}{3}, \mathrm{Sub}_2^R{:}\frac{1}{3}, \mathrm{Sub}_2^S{:}\frac{1}{3}\}$, yielding three execution paths that `check_with_dfa_scenic` evaluates by importance sampling over a single shared per-primitive trace pool.

Specification. The DFA monitor is the flat-safety `safe_under_max` entry of Table 2: a two-state automaton with start ok, accepting set {ok}, and absorbing bad. The per-row labeling function emits high when the post-warmup speed exceeds $5.5\,\mathrm{m/s}$ and low otherwise, with `WARMUP_STEPS` $= 25$.

Leaf primitives. Sub_1 is the cold-start approach segment and is run without a prewarm prefix. Each of $\mathrm{Sub}_2^L, \mathrm{Sub}_2^R, \mathrm{Sub}_2^S$ executes a `FollowTrajectoryBehavior` along the left, right, or straight maneuver through the intersection and is preceded by a randomized cruise prewarm so that step 0 of its trimmed trace represents a warm state matching the exit distribution of Sub_1 (same mechanism as in Set B; see §A.2).

Trace generation. The compositional simulation spawns four parallel `_worker_generate_scenario` processes, one per primitive, each writing an incremental CSV. Per-trace simulation horizons are `MAX_STEPS` $=$ 85 for Sub_1 and `MAX_STEPS` $= 110$ for each $\mathrm{Sub}_2^{\{L,R,S\}}$, where the extra 25 rows cover the prewarm prefix that is trimmed after generation. The monolithic simulation runs a single worker on `MonolithicMain`, the chained-behavior counterpart that executes the approach followed by a uniformly sampled turn back-to-back, at `MAX_STEPS` $= 170$ ticks per trace.

Wall-clock protocol. Each method runs one continuous simulation up to $T_{\max} = 1800\,\mathrm{s}$ and records per-primitive trace counts every $\Delta T = 30\,\mathrm{s}$, yielding up to 60 snapshot points per method. After the run, for every snapshot $(T_i, \{n_i^{(s)}\}_s)$ the harness materializes filtered CSVs containing the first $n_i^{(s)}$ trace IDs per primitive, applies the DFA labeling, and evaluates $(\hat{\rho}, \hat{\varepsilon})$ via `check_with_dfa_scenic`. This produces the full $(T, \hat{\rho}, \hat{\varepsilon})$ trajectory of each method from a single simulation per method rather than one independent run per budget level. Total wall-clock cost is $2\,T_{\max} = 3600\,\mathrm{s}$, sequential across the two methods.

Reading the figure. Checkpoints with fewer than two traces per primitive on the compositional side cannot be evaluated by the importance-sampling estimator (the KDE fit degenerates) and are recorded as `insufficient_data` rows in the result CSV; these are skipped at plotting time, which is why the compositional curve in Fig. 4 begins at $T \approx 90\,\mathrm{s}$ while the monolithic curve begins one snapshot earlier. Past $T = 90\,\mathrm{s}$, both curves are populated at every checkpoint. The ratio $\hat{\varepsilon}_{\mathrm{mono}}(T)/\hat{\varepsilon}_{\mathrm{comp}}(T)$ lies between 4 and 7 across the measured budgets, consistent with the 4-way primitive parallelism and the shorter per-primitive trace length.

B Bounded Liveness via Safety Complement: a Worked Example

We empirically validate the complement trick discussed in §5 on a Scenic-defined 4-way intersection composite. Because MetaDrive traces are finite (bounded by T ticks per primitive), the liveness property below is implicitly bounded: "eventually" means "at some step within the trace." The composite is

$$\mathsf{Sub1}\,;\,\mathsf{choose}\{\mathsf{Sub2L},\mathsf{Sub2R},\mathsf{Sub2S}\}$$

The ego first traverses an approach segment, then takes one of three turns (left, right, straight) drawn uniformly at the intersection. We generate N=300 MetaDrive traces per primitive (with a 25-tick prewarm trim on the post-intersection segments to align KDE handoff distributions) and 300 traces of the monolithic composite for ground truth.

Bounded liveness → safety reduction. The target bounded-liveness property is

$$\varphi_{\text{live}} \;\equiv\; \Diamond_{\le T}\big(\text{speed} > 5.5\ \text{m/s}\big) \quad \text{post-warmup},$$

i.e. "the vehicle reaches high speed at some point within the trace." Its negation is the flat safety property

$$\varphi_{\text{safe}} \;\equiv\; \Box_{\le T}\big(\text{speed} \le 5.5\ \text{m/s}\big) \quad \text{post-warmup},$$

encoded as a 2-state DFA with states $\{\mathsf{ok},\mathsf{bad}\}$, start ok, accepting set $\{\mathsf{ok}\}$, and bad absorbing. Note that because the DFA has only two states and bad is absorbing, φ_{safe} is in fact Markovian on the raw simulation state; this is why a single safety monitor suffices. The estimator computes $\hat{\rho}_{\text{safe}}$ via the standard compositional procedure of §3.2, and we recover $\hat{\rho}_{\text{live}} = 1 - \hat{\rho}_{\text{safe}}$.

Table 3. Bounded liveness estimated via safety complement on the 4-way intersection composite. Compositional uses only per-primitive trace pools (N=300 each); monolithic re-simulates the full composite. Reported intervals are Hoeffding bounds at $\delta = 0.05$ for the monolithic estimator and the self-normalized importance-sampling standard error for the compositional estimator. The two agree to $|\Delta\rho| = 0.020$, well inside both intervals. Per-primitive safety rates: Sub1=0.743, Sub2L=0.527, Sub2R=0.563, Sub2S=0.580.

	Compositional	**Monolithic**
$\hat{\rho}_{\text{safe}}$	0.4138 ± 0.0601	0.3933 ± 0.0784
$\hat{\rho}_{\text{live}} = 1 - \hat{\rho}_{\text{safe}}$	0.5862 ± 0.0601	0.6067 ± 0.0784

Discussion. Table 3 confirms that the complement trick is sound when the liveness property is bounded and its negation is a flat safety DFA: the compositional pipeline (including the random-branch KDE mixture at the choose node) transfers verbatim to the safety complement, and the recovered bounded-liveness estimate matches the monolithic ground truth within the reported uncertainty. The DFA's two-state structure is precisely what makes this work: the negation collapses to a flat safety monitor, so the absorbing-reject machinery applies. For the bounded-response, bounded-persistence, and bounded-recurrence patterns enumerated in §5, the negation contains nested temporal operators and the corresponding DFA has more than one "live" region; no single absorbing-reject relabeling exists, and the compositional estimator as defined here does not apply.