

SilGAN: Generating driving maneuvers for scenario-based software-in-the-loop testing

Dhasarathy Parthasarathy^{†*}, Anton Johansson^{*}

[†]Volvo Group, Gothenburg, Sweden, Email: dhasarathy.parthasarathy@volvo.com

^{*}Chalmers University of Technology, Gothenburg, Sweden, Email: johaant@chalmers.se

Abstract—Automotive software testing continues to rely largely upon expensive field tests to ensure quality because alternatives like simulation-based testing are relatively immature. As a step towards lowering reliance on field tests, we present SilGAN, a deep generative model that eases specification, stimulus generation, and automation of automotive software-in-the-loop testing. The model is trained using data recorded from vehicles in the field. Upon training, the model uses a concise specification for a driving scenario to generate realistic vehicle state transitions that can occur during such a scenario. Such authentic emulation of internal vehicle behavior can be used for rapid, systematic and inexpensive testing of vehicle control software. In addition, by presenting a targeted method for searching through the information learned by the model, we show how a test objective like code coverage can be automated. The data driven end-to-end testing pipeline that we present vastly expands the scope and credibility of automotive simulation-based testing. This reduces time to market while helping maintain required standards of quality.

Index Terms—software-in-the-loop testing, generative adversarial networks, time series generation, latent space search

I. INTRODUCTION

Embedded software lies at the heart of a variety of internal vehicle functions such as engine control, driver assistance, and energy management [1]. As modern software-driven commercial vehicles grow more capable, they are becoming indispensable parts of a variety of industrial activities such as urban distribution, public transport, construction, and mining. This, however, creates a growing demand for new functionality, placing increased pressure on vehicle manufacturers to rapidly develop, test, and deploy new software. Increasing the pace of development, while continuing to deliver quality software, are urgent priorities that the automotive industry faces today [2]. One way in which the industry has responded is by taking a platform approach. Today, it is typical for core applications like driveline or battery management to be re-used across different vehicle platforms. While this streamlines engineering effort, the applications themselves begin to be used in a variety of different ways. This inevitably increases the chance that they regress when being used in poorly understood or unanticipated ways. For example, electrically steered axles of a truck could draw higher than expected current only when it rolls downhill at an inclination and surface friction that is common in pit mines. In trying to identify and avoid such failures, vehicle manufacturers typically spend significant effort, subjecting software to thousands of kilometers worth of time consuming field tests.

Recognizing the clear need to accelerate test feedback, the industry has been turning more towards simulation-based testing with software-in-the-loop (SIL) [3]. The vehicle software system consists of Software Components (SWCs), distributed over electronic control units, which exchange information using *signals*. Each signal captures one element of vehicle state, such as the current speed or steering angle. SWCs realize their functions by responding to transitions in signals which are routed to them. Therefore, for credible simulation-based testing with SIL, especially for low-level control software, test setups need to generate signal transitions at a deep level of detail. Today's SIL tests rigs simulate these transitions by manually specifying explicit, domain specific, mathematical rules in the form of plant models. This works well when testing individual SWCs, where plant models need to simulate transitions of only a handful of signals. As the scope expands to testing larger sub-systems of SWCs, under the influence of realistic driving, modeling takes significant effort. This is because, during a driving maneuver that lasts several minutes, the continuous, multi-dimensional vehicle state-space undergoes complex transitions, many of which are difficult to model.

However, the ongoing data revolution in transportation means that vehicle manufacturers now have unprecedented insights into how their vehicles are being used in the field. The availability of internal vehicle data, combined with rapid advances in deep learning, makes it possible to learn intricate state transitions that occur during different driving scenarios. Leveraging techniques of deep generative modeling, in this work, we contribute the following (refer Figure 1).

- We introduce *templates* as a simple, concise and granular specification of a driving scenario
- We train a model to translate a template into realistic multi-dimensional signal transitions. Transitions, so generated, emulate authentic vehicle behavior during the specified scenario and can be used as stimulus for software under test.
- For given software under test, we show how stimulus generation can be automated to satisfy a specific test objective,

This novel set of contributions significantly expand the credibility of simulation-driven automotive software testing, further lowering the reliance on field tests. This reduces time and cost to market, while helping maintain the necessary standards of quality. Code from this work is publicly available¹.

¹<https://github.com/dhas/silGAN>

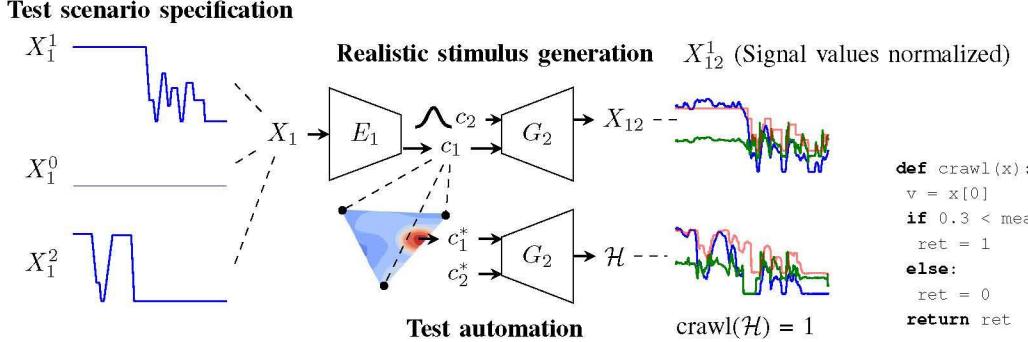


Fig. 1: Overview of SilGAN - A test scenario (Section II) of a vehicle coming to a stop can be specified using template X_1^1 , which is a rough sketch of how the vehicle speed is expected to vary. We train SilGAN (Section III) to translate this template into a maneuver X_{12}^1 , containing a realistic rendition of the template along with related transitions of engine speed and selected gear. Such a generated realistic maneuver can be applied as stimuli for software under test. The test scenario can also be arbitrarily enriched by including conditions like template X_1^2 , depicting another variation of a vehicle stopping. Generated maneuvers can thus be constrained to combine only the characteristics of specified templates (Section IV). Having defined such a composite scenario, we show how SilGAN can be used for test automation (Section V). Consider *crawl*, a simple Python function that checks whether a vehicle is crawling at a relatively low speed. Using feedback from a transformed version of *crawl*, we demonstrate a method that searches through the model’s latent information spaces to automatically generate a realistic stop maneuver \mathcal{H} . Generated test stimulus \mathcal{H} not only complies with the specified test scenario, but also makes $crawl(\mathcal{H}) = 1$.

II. TEST SCENARIOS FOR CONTROL SOFTWARE

As surveyed in [4], recent years have seen many proposals that help specify scenarios for testing automotive software systems. However, a major limitation that they share is a focus on specifying scenarios only in terms of high-level driving characteristics. For example, popular ontologies like [5] describe driving scenarios mostly in terms of tactical aspects like avoiding obstacles or changing lanes. Such proposals may have been used to analyze and test driver assistance or autonomous driving controllers (for example, [6]), but they define scenarios at too low a level of granularity. Vehicle control software at the operational level, i.e. those for functions like fuel-injection or automatic gear shifting, react to transitions of a multitude of internal vehicle signals. Testing these controllers under different driving scenarios would therefore require realistic specification of these signals at a granular level of detail. The current practice of manual plant modeling addresses this to a certain extent. But, when the scope of testing widens or the time-horizon of testing extends beyond a few seconds, plant models need support in capturing complex, non-linear and unforeseen internal vehicle phenomena.

A. Maneuver - a multi-dimensional time series of signals

Vehicles often execute recognizable driving patterns, and we refer to a family of similar patterns as a driving *scenario*. One example of a driving scenario would be takeoff, where the vehicle starts rolling and subsequently begins to cruise. Depending upon a variety of factors like vehicle mass, road inclination, surface and traffic conditions, there are several ways of executing takeoff. We refer to one specific execution of a driving scenario as a driving *maneuver*. Figure 2a captures ~ 10 -minute long maneuvers of takeoff and cruise scenarios using three recorded signals vehicle speed, engine speed and

selected gear. Takeoff, for instance, is interesting because during its execution, apart from several core control actions, additional ones like automatic engine start, cabin heat circulation, and parking brake release are activated. Testing the behavior of such functions under many possible instances of takeoff can therefore be of great value. However, as noted earlier, if one were to collectively test SWCs for all these controllers, then detailed signal transitions on the form shown in Figure 2a must be applied as test input. With numerous possible variations of takeoff, manually specifying transitions of several signals, for even a handful of takeoff maneuvers, is clearly difficult.

B. Template - a 1-D signal-level scenario description

The difficulty in specifying a realistic test maneuver arises from two problems - (i) the need to specify detailed short and long term transitions of each constituent signal, and (ii) capturing possible inter-signal dependencies. What would help, therefore, is a specification mechanism that ignores details and focuses only on the basic profile of the maneuver. This can be followed by another mechanism that can translate this profile into a realistic maneuver. Building upon previous work for test stimulus generation [7], we propose piece-wise linear approximation as a simple way to avoid specifying transient details of a signal. The resulting *template*, as shown in Figure 2b, serves as a rough, intuitive sketch of the long-term signal profile. To address the specification of inter-signal dependencies, one could use the proposal in [7] to jointly specify a template for each signal included in the driving maneuver. In practice, however, specifying plausibly related templates for all signals is quite difficult. We therefore propose a much simpler approach of specifying a driving scenario by sketching a template for *any one* constituent signal. This completely eliminates the need to address dependencies. As

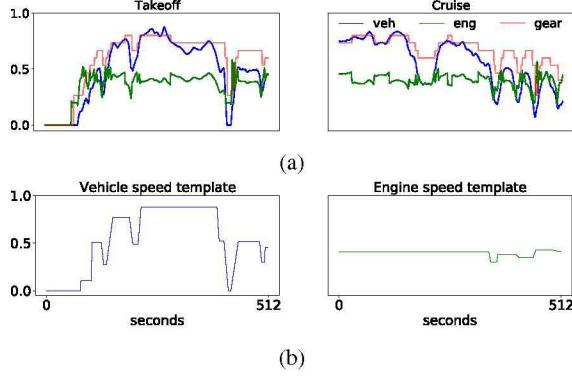


Fig. 2: Recorded driving maneuvers (a) and corresponding templates (b). Signal values normalized

seen in Figure 2b, this means that a tester can choose to define a takeoff scenario by specifying only a vehicle speed template. The same tester can choose to define a cruise scenario by only specifying a template for the engine speed. This flexibility allows test case design to focus on a single chosen control signal, further reducing the specification effort. By thus specifying long-range driving scenarios at level of detail suitable for testing entire sub-systems of SWCs, templates are also different from alternative proposals like [8], which focus on specifying test scenarios for individual software routines.

Templates may simplify the specification of a test scenario for low-level control software, but translating them into realistic maneuvers is not straightforward. Not only should such translation restore the details of the sketched signal, but it should also generate realistically inter-related accompanying signals. To achieve this, we train a deep generative model.

III. SILGAN - TRANSLATING TEMPLATES TO MANEUVERS

Let us define the domain of one-dimensional templates as \mathcal{X}_1 , and that of L -dimensional driving maneuvers as \mathcal{X}_2 . Let $P_l(X_1)$ denote the distribution of templates for the l^{th} signal of the L -dimensional system, and $P(X_2)$ denote the distribution of maneuvers. Denoting the duration of templates and maneuvers by N , a template $X_1 \sim P_l(X_1)$, is a time-series in $\mathbb{R}^{1 \times N}$, while a maneuver $X_2 \sim P(X_2)$ is a time-series in $\mathbb{R}^{L \times N}$. The objective is to learn $\mathcal{T} : \mathcal{X}_1 \rightarrow \mathcal{X}_2$ that can map X_1 into a realistic maneuver $X_{12} \sim P_l(X_2|X_1)$, that is faithful to the specified profile in the l^{th} signal and generates plausible related variations for all $(L - 1)$ other signals. Deep learning techniques that achieve such mappings by implicitly estimating the conditional distribution fall under the general category of deep domain adaptation [9].

A. Training data

The number of signals included in a driving maneuver clearly plays an important role in defining its scope for testing. Here, to sufficiently showcase the capabilities of the model, we fix $L = 3$ and $N = 512$. The training data comes from a source of signals recorded at 1 Hz from 19 Volvo buses operating over a 3-5 year period. From this around 200k, 512-second

long snapshots of three important signals related to vehicle dynamics – vehicle speed, engine speed, and selected gear – have been collected for training. Selected signals exhibit a good mix of levels of granularity. Vehicle speed is a high-level characteristic, while engine speed and (automatically) selected gear are detailed characteristics of the driveline. Moreover, these signals are routed to a wide variety of SWCs, making them relevant for a number of SIL test cases. Translating a template into a maneuver may be complex but, given a recorded maneuver X_2 , piecewise linear templates $X_1^{(L)} = (X_1^1, \dots, X_1^L)^T$ for all L signals can be automatically extracted using relatively simple procedures. Upon smoothing a signal using windowed mean and 1-D Sobel filtering, its template is approximated as flat regions around major points inflection, with straight-line edges connecting these regions (refer Figure 2). The resulting training data therefore consists of pairs $(X_1^{(L)}, X_2)$ of templates $X_1^l \sim P_l(X_1^l)$ and recorded maneuvers $X_2 \sim P(X_2)$ they were extracted from. In using this training set, it is important to note our assumption that templates specified by a tester can be plausibly sampled from the distribution of extracted templates $P_l(X_1^l)$. Since a template X_1^l is always associated with l , the signal which it approximates, for ease of notation, we cease associating it, or any of its mappings, explicitly with l .

B. Model design

Translation - A translated maneuver X_{12} is expected to realistically render the template X_1 as its l^{th} signal. Since X_1 and X_{12} clearly share some characteristics, let us define C_1 as a latent intermediate domain that encodes this shared information. Let $E_1 : \mathcal{X}_1 \rightarrow C_1$ be a learnable encoder that maps a template into this domain. However, the translated maneuver X_{12} is also required to render realistic transitions of all other signals. Let us therefore define another latent domain C_2 as the source of all this additional information. Allowing the learning process to optimally structure its information, codes from this domain are sampled as $c_2 \sim \mathcal{N}(0, I)$. The translation process can then be completed by defining a learnable generator $G_2 : C_1 \times C_2 \rightarrow \mathcal{X}_2$, that produces a maneuver by combining information from both latent domains (1). In order to ensure that the translated maneuver is realistic, a least-squares discriminator [10] $D_2 : \mathcal{X}_2 \rightarrow \mathbb{R}$ is defined. Using ground truth labels 0 for fake samples and 1 for real samples, it learns to estimate the conditional distribution $P_l(X_2|X_1)$, and feeds back stably minimizable criticism of generated samples. The complete forward mapping generator can be learnt by minimizing losses (2) and (3) in an adversarial fashion [11].

$$X_{12} = G_2(E_1(X_1), c_2), \quad c_2 \sim \mathcal{N}(0, I) \quad (1)$$

$$\mathcal{L}_{\text{Gen}}^{\text{Tran}} = \mathbb{E}_{X_1, c_2} (D_2(X_{12}) - 1)^2 \quad (2)$$

$$\mathcal{L}_{\text{Dis}}^{\text{Tran}} = \mathbb{E}_{X_2} (D_2(X_2) - 1)^2 + \mathbb{E}_{X_1, c_2} D_2(X_{12})^2 \quad (3)$$

$$\mathcal{L}_{\text{Pair}}^{\text{Tran}} = \mathbb{E}_{X_1, c_2, X_2} \|X_{12} - X_2^l\|_1 \quad (4)$$

$$X_1 \sim P_l(X_1), \quad X_2 \sim P(X_2)$$

The adversarial loss is normally sufficient to ensure both translation into a realistic maneuver and adherence to the profile in the template. However, we find that this

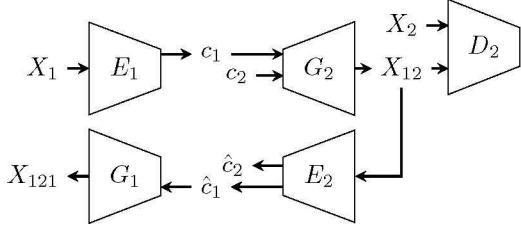


Fig. 3: SilGAN - Translation stage

adherence is sensitive to random seeding. To reliably ensure adherence while translating template X_1 , a pairing loss (4) is used to encourage reconstruction of the l^{th} signal of the corresponding real maneuver X_2 . Also, (1) shows that upon encoding a template into c_1 , G_2 can combine it with different codes c_2 to produce different maneuvers, all of which satisfy the profile requirements set by c_1 . Such a technique of defining disentangled, partially shared, information domains for multi-modal domain translation, was first proposed in [12].

Prior work has shown that the quality of domain translation can be improved by encouraging cycle-consistency [13]. That is, having translated template X_1 into a maneuver X_{12} , it is beneficial to reverse the translation and recover the template (5). We therefore define functions $E_2 : \mathcal{X}_2 \rightarrow C_1 \times C_2$ and $G_1 : C_1 \rightarrow \mathcal{X}_1$ that reverse the translation. Unlike template extraction procedures used in preparing the training set which achieve the same end, reverse translation (5) is differentiable. This allows the cycle consistency objective (6), the L_1 loss between input and recovered templates, to back propagate gradients from the reverse to the forward process during training.

$$X_{121} = G_1(\hat{c}_1), \hat{c}_1, \hat{c}_2 = E_2(X_{12}) \quad (5)$$

$$\mathcal{L}_{Cyc}^{Tran} = \mathbb{E}_{X_1, c_2} \| X_{121} - X_1 \|_1 \quad (6)$$

Forward and cycle-translation phases collectively constitute the translation stage (Figure 3) of SilGAN. Networks in this stage use 1-D fully-convolutional layers, and can process templates of a duration of $N \leq 512$, as long as N is a power of 2.

Expansion - One limitation of the translation stage is that it maps a template of duration N into a maneuver of the same duration. Generating long maneuvers can therefore be cumbersome, since a template needs to be specified for its entire timeline. This is alleviated by defining an expansion stage $\mathcal{E} : \mathcal{X}_2 \rightarrow \mathcal{X}_3$ (Figure 4). Here, domain \mathcal{X}_3 comprises maneuvers $X_3 \sim P(X_3)$ of duration $M > N$. A common method for expanding a time series is forecasting. However, typical forecasting (backcasting) focuses only on the future (past) of a given series. The goal of expansion is different since it requires the generation of future and past transitions, both of which are jointly related to the given series. Here, X_{12} is expanded into a longer maneuver $X_{13} \sim P_l(X_3|X_1)$, by jointly generating plausible preceding (F_1) and succeeding (F_2) transitions for all L signals. A learnable generator $G_3 : \mathcal{X}_2 \times C_3 \rightarrow \mathcal{F}_1 \times \mathcal{F}_2$, where the latent domain C_3 acts as the source of expanded information, generates extra parts (7) of length $(M - N)$ each. This ensures that cropping the con-

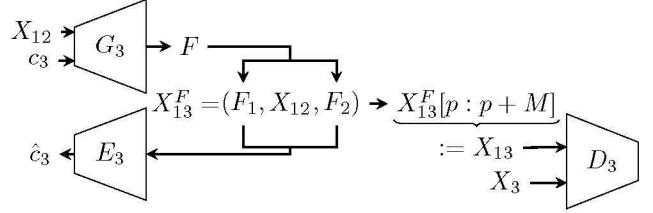


Fig. 4: SilGAN - Expansion stage

catenation X_{13}^F according to (8) makes X_{12} appear anywhere within X_{13} . Preceding and succeeding parts flexibly occupy the rest of the timeline. To ensure that expanded maneuvers are plausibly realistic, an additional least-squares discriminator $D_3 : \mathcal{X}_3 \rightarrow \mathbb{R}^+$ is defined. The function G_3 learns to produce realistic expansions in concert with D_3 , by minimizing a novel *random cropping adversarial loss*. This simple loss, defined by (9) and (10), operates by randomly cropping the concatenation X_{13}^F using $p \sim \mathcal{U}\{0, M - N\}$ before evaluating it with D_3 . This ensures that any crop of duration M is realistic.

$$X_{13}^F = (F_1, X_{12}, F_2), F_1, F_2 = G_3(X_{12}, c_3) \quad (7)$$

$$X_{13} = X_{13}^F[p : p + M] \quad (8)$$

$$\mathcal{L}_{Gen}^{Exp} = \mathbb{E}_{X_1, c_2, c_3, p} (D_3(X_{13}) - 1)^2 \quad (9)$$

$$\mathcal{L}_{Dis}^{Exp} = \mathbb{E}_{X_3} (D_3(X_3) - 1)^2 + \mathbb{E}_{X_1, c_2, c_3, p} D_3(X_{13})^2 \quad (10)$$

$$X_3 \sim P(X_3), c_2, c_3 \sim \mathcal{N}(0, I), p \sim \mathcal{U}\{0, M - N\}$$

To showcase expansion, we fix $M = 1024$ so that any template of duration $N \leq 512$, as long as it is a power of two, can be expanded to a maneuver of duration 1024. To train the expansion stage, a separate set of recorded maneuvers of this duration is collected. While networks in the translation stage use 1-D convolution, 2-D convolution is found to be more effective in expansion. With an increased duration, stacking up parts of the timeline and applying a 2-D kernel seems more effective.

Bidirectional reconstruction - Upon observing the layout of either stage, one can note the presence of autoencoders $E_i((G_i))$, $i = 1, 2, 3$. The learning process can therefore be eased using objectives (11), which encourage them to learn identity mapping in the respective domains. This, however, does not apply to the autoencoder in the expansion stage, since it does not operate directly on the domain \mathcal{X}_3 . In addition, as shown in [12], learning identity mappings in the code domains using objectives (12) and (13), is a simple way of encouraging code distributions to match $\mathcal{N}(0, I)$.

$$\mathcal{L}_{ID_i} = \mathbb{E}_{X_i} \| X_i - G_i(E_i(X_i)) \|_1, i = 1, 2 \quad (11)$$

$$\mathcal{L}_{CR_2} = \mathbb{E}_{X_1, c_2} \| (c_1, c_2) - E_2(G_2(c_1, c_2)) \|_1 \quad (12)$$

$$\mathcal{L}_{CR_3} = \mathbb{E}_{X_1, c_2, c_3} \| c_3 - E_3(G_3(X_{12}, c_3)) \|_1 \quad (13)$$

$$c_1 = E_1(X_1^l), c_2, c_3 \sim \mathcal{N}(0, I)$$

This is not only important to ease sampling, but also essential for generalizing the models generative capabilities. Such a simple code reconstruction objective avoids complex alternatives like Kullback-Leibler loss, while also, as empirically observed, producing translations of a better quality.

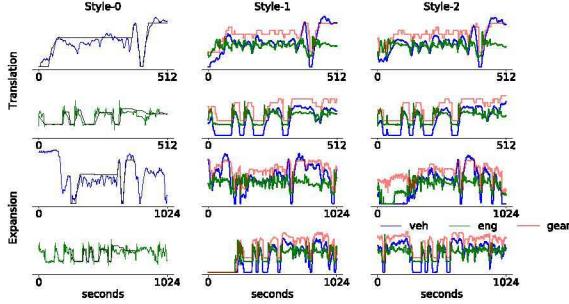


Fig. 5: Generation of maneuvers by specifying one vehicle speed and one engine speed template. The first column shows the template overlaid on the corresponding generated signal.

End-to-end training - Combining all training objectives previously defined, SilGAN is trained end-to-end by minimizing the composite adversarial objective (14).

$$\begin{aligned} \mathcal{L}_{Gen} &= \min_{E_1, E_2, G_1, G_2} \lambda_{Gen} (\mathcal{L}_{Gen}^{Tran} + \mathcal{L}_{Gen}^{Exp}) + \lambda_{Pair} \mathcal{L}_{Pair}^{Tran} \\ &+ \lambda_{Cyc} \mathcal{L}_{Cyc}^{Tran} + \lambda_{ID} \sum_{i=1,2} \mathcal{L}_{ID_i} + \sum_{j=2,3} \lambda_{CR} \mathcal{L}_{CR_j} \\ \mathcal{L}_{Dis} &= \min_{D_2, D_3} (\mathcal{L}_{Dis}^{Tran} + \mathcal{L}_{Dis}^{Exp}) \\ \mathcal{L} &= \mathcal{L}_{Gen} + \mathcal{L}_{Dis} \end{aligned} \quad (14)$$

Results shown here are sampled from a model trained with relative importance for all objectives set to 1, except $\lambda_{ID} = 10$. Further details about training are available in the released code. Depending upon the configuration, training the model takes 5–15 minutes per epoch on a single NVidia Tesla V100 GPU. To evaluate the quality of generation, a held-out test set of $\sim 20k$ recorded maneuvers, each of duration 512s, and corresponding extracted templates is used. As an end-to-end indication of the quality of translation, cycle reconstruction of the test set is measured using (6). However, instead of L_1 distance, the easier to interpret Structural Similarity Index (SSIM) is used. SSIM between two time series is a real number in $[0, 1]$, with 0 indicating dissimilarity and 1 indicating a perfect match. Cycle reconstruction is measured as the average SSIM between a test template X_1 and 4 cycle translations X_{121} , each using a different random code c_2 . After training for ~ 15 epochs, cycle reconstruction SSIM, averaged over the entire test set, settles around 0.95. While such high similarity of cycle reconstruction indicates healthy training, quality of forward translation is additionally ensured by visually inspecting around hundred randomly selected test template translations. Visual inspection of time series maneuvers confirms adherence to the template and plausibility of characteristic features like correlations between vehicle and engine speed under the influence of gear shifts.

IV. USING SILGAN FOR TEST STIMULUS GENERATION

Upon training, simple scenario-based generation of realistic driving maneuvers is accomplished by specifying a single template. As shown in Figure 5, a template specified for any one constituent signal can be translated and expanded into diverse, but realistic, interpretations. Here, the choice

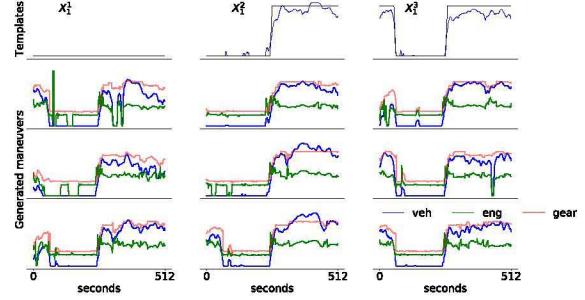


Fig. 6: By specifying three templates (top row) – null, takeoff and stop before takeoff, generating maneuvers using (15) are constrained to combine characteristics of all three

$p = \frac{M-N}{2}$ ensures that the translated maneuver appears at the center of the expanded timeline. It can be seen that translations are quite faithful in adhering to the template, while still showing subtle variations in its details, across interpretations. The expanded maneuvers, on the other hand, show substantial diversity in preceding and succeeding signal transitions.

A more potent way of using SilGAN would be with multiple templates, ensuring compliance with a combination of scenarios. It is well-known [14] that generative models achieve non-linear combination of characteristics in the sample space using trivial linear combinations in its latent space. As noted earlier, systematically generating a variety of takeoff maneuvers is valuable for testing critical functions, and Figure 6 shows how this can be easily achieved. Let us start by specifying a helpful *null* vehicle speed template, X_1^1 , where the vehicle is completely still for 512 seconds. Let us then specify one takeoff template X_1^2 where the vehicle stays still for half that time before beginning to roll. Since this may be considered too narrow a takeoff scenario, let us widen it by specifying an additional template X_1^3 , where the vehicle stops once before taking off. The triple of codes $c_1^{(3)} = (E_1(X_1^1), E_1(X_1^2), E_1(X_1^3))^T$ can be imagined as the vertices of a triangle, with each vertex being embedded in the high-dimensional latent domain C_1 . Sampling any point on this triangle, along with a random code c_2 , and decoding them using G_2 , guarantees that generated maneuvers do not stray beyond the driving conditions collectively posed by the three chosen templates (Figure 6). Such constrained, but realistic, generation produces authentic stimuli for SIL testing under a controlled composite test scenario. This extends to the general case of using K templates, meaning that it is possible to generate maneuvers that is guaranteed to comply with a scenario of an arbitrary level of sophistication. Sampling in the latent hyper-plane with K vertices is achieved by drawing a simplex $\alpha^{(K)}, \sum_{i=1}^K \alpha_i^{(K)} = 1$, from a Dirichlet distribution of order K , and linearly combining it with codes $c_1^{(K)}$ (15). Repeatedly drawing simplexes also ensures that the hyper-plane is uniformly covered, making it an apt tool for exploring the variety of combinations that are possible.

$$\begin{aligned} \alpha^{(K)} &\sim Dir(K, \mathbf{1}^K), c_1^{(K)} = (E_1(X_1^1), \dots, E_1(X_1^K))^T \\ X_{12} &= G_2(\alpha^{(K)} \cdot c_1^{(K)}, c_2), c_2 \sim \mathcal{N}(0, I) \end{aligned} \quad (15)$$

```

def agg_to(v, e):
    m1 = mean(v[256:350])
    m2 = mean(v[350:512])
    m3 = mean(e[0:256])
    if(0.49 < m1 < 0.51) &
        (0.59 < m2 < 0.61) &
        (m3 < 0.08):
        ret1 = 1
    else:
        ret1 = 0
    return ret1

def S(v, e):
    m1 = mean(v[256:350])
    m2 = mean(v[350:512])
    m3 = mean(e[0:256])
    c0 = and(lt(0.49, m1),
              lt(m1, 0.51))
    c1 = and(lt(0.59, m2),
              lt(m2, 0.61))
    c2 = lt(m3, 0.08)
    c = and(and(c0, c1), c2)
    return c

```

(a)

(b)

Fig. 7: A simple function (a) in software under test which is automatically transformed into a search function (b) that returns one coverage indicator per branching condition

V. USING SILGAN FOR TEST AUTOMATION

Let us choose templates $X_1^{(3)} = (X_1^1, X_1^2, X_1^3)^T$, described previously, as a test scenario for takeoff. In the software under test, let us assume there is function *agg_to* (Figure 7a) that observes vehicle and engine speed signals to check whether the engine largely idles before the vehicle quickly takes off and cruises at increasing speeds. One can find such a function as part of logic that identifies aggressive driving events. Events, so identified, are either logged for analysis or are actively compensated, for example, by adjusting fuel consumption or exhaust after treatment. Under the takeoff test scenario $X_1^{(3)}$, let the objective be to find a maneuver \mathcal{H} that checks the design of takeoff defined by *agg_to*. This can be seen as a form of code coverage, i.e. finding a stimulus that satisfies the assumptions in the *if* condition that appears in the test function. While simplex sampling using (15) may be effective for generating a variety of maneuvers that comply with the defined scenario, the number of possibilities is infinite. Finding \mathcal{H} by random sampling is therefore an impractical search technique. More importantly, *agg_to* indicates coverage solely by returning 1 when the *if* condition is satisfied. It therefore poses a fundamental problem in providing no real-time feedback to guide the sampling process.

Code coverage entails satisfying branching conditions, each of which is typically defined using a composition of boolean operations. Since these operations evaluate to only *true* or *false*, there is no indication of how distant a test input is from making an *if* condition evaluate to, say, *true*. However, cases like the chosen test function, where branching conditions evaluate real values, are fairly routine in decision making logic found in vehicle control software. In such cases, it is possible to convert discrete boolean operations into real-valued measures. Setting aside boolean operators $=$ and \neq which are not directly applicable to real values, atomic operations $<$ and $>$ can be readily converted into difference functions (16). Arguments that evaluate to a large positive value indicate significant mismatch, and those that evaluate to a negative value indicate a certain match for both conditions. This also means that \neg , i.e. guaranteeing the mismatch of an atomic condition, is simply

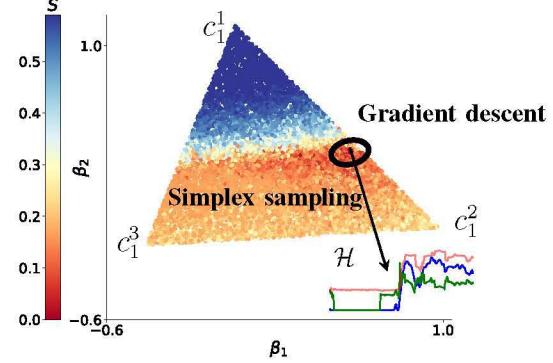


Fig. 8: With test scenario $X_1^{(3)}$, automatic code coverage for the test function in Figure 7 begins with simplex sampling in the triangle with vertices $c_1^{(3)}$ in the latent space C_1 . Once a promising area is found, gradient-descent takes over to find \mathcal{H} that satisfies the condition. For visualization in 2-d, we plot the triangle by taking steps β_1, β_2 along two randomly chosen basis vectors that span its plane. A similar search is jointly conducted in C_2 , but it is not visualized here.

achieved by arithmetic negation². This principle extends to \wedge and \vee , where functions *max* and *min* respectively check whether all, or any, atomic operation(s) evaluate to a negative value. While alternative definitions are certainly possible, chosen real-valued equivalents of boolean operations (17) are not only simple but also differentiable³. One simple check of consistency is that these functions satisfy De Morgan's laws⁴.

$$\text{lt}(a, b) = a - b \quad \text{gt}(a, b) = b - a \quad (16)$$

$$\text{and}(c, d) = \max(c, d), \quad \text{or}(c, d) = \min(c, d) \quad (17)$$

$$\text{not}(c) = -c$$

In transforming boolean operations on real values into another real value, these functions give continuous feedback on how distant a test input is from covering a branching condition on real values. We therefore refer to them as *coverage indicators*. By traversing its abstract syntax tree, substituting boolean operators with coverage indicators, and extracting a composite indicator for its *if* condition, the chosen test function is automatically transformed into a search function $S : \mathcal{X}_2 \rightarrow \mathbb{R}$ (Figure 7b). This means $S < 0$ guarantees that the corresponding input maneuver is a \mathcal{H} that satisfies the branching condition. Further, since the coverage indicators are readily differentiable, as long as operations leading up to branching conditions are also differentiable, the resulting S is itself differentiable. This means that the right combination of $\alpha^{(N)}$ and c_2 , that generates \mathcal{H} , can be found using gradient descent.

While the search function may be differentiable, its multi-dimensional landscape is likely to feature many local minima.

²Defining *not* as arithmetic negation restricts it to a subset of possible solutions. For example $\neg(a < b) = a \geq b$, but *not* only finds solutions $a > b$, setting aside equality checks on real values

³ $\max(c, d)$ and $\min(c, d)$ are not differentiable at $c = d$, but this can be disregarded for real-valued c and d

⁴ $\text{not}(\text{and}(a, b)) = -\max(a, b) = \min(-a, -b) = \text{or}(\text{not}(a), \text{not}(b))$

As seen in Figure 8, the presence of large flat regions, where S hardly changes, clearly hinders search by gradient descent. To address this issue, we propose a novel search method that *combines sampling and gradient-descent* to automatically generate test stimuli that match the design of code under test (Algorithm 1). This method begins with simplex sampling for n_{sim} iterations in order to survey the coverage landscape in the triangle $c_1^{(3)}$. If the sampling process chances upon \mathcal{H} , the objective is achieved, ending the search. Otherwise, the algorithm identifies a promising area, i.e. a combination of α^* and c_2^* that generates a maneuver resulting in $S \approx 0$, around which finding a \mathcal{H} is likely. The search is then taken over by gradient descent, which attempts to iteratively minimize S by jointly varying α^* and c_2^* along directions of fastest decrease in S . Reparameterizing α^* using logit/sigmoid ensures that gradient-descent does not stray outside the triangle, guaranteeing compliance with the test scenario. This search persists for a maximum of n_{gd} steps during which, if S becomes negative, \mathcal{H} has been found and the search is complete. Otherwise, the search times out without finding \mathcal{H} under the given test scenario.

Algorithm 1: Scenario-based test automation

Input	: Scenario of templates $X_1^{(K)}$, search function S
Output	: Maneuver \mathcal{H} satisfying the coverage condition
Parameters	: n_{sim} - max sampling steps, n_{gd} - max gradient descent steps, η - gradient step size

```

1  $c_1^{(K)} = (E_1(X_1^0), \dots, E_1(X_1^K))^T$ 
2  $s_{min} = \infty$ ,  $\alpha^* = null$ ,  $c_2^* = null$ 
3 for  $n \in [n_{sim}]$  do
4    $\alpha^{(K)} \sim Dir(K, \mathbb{1}^K)$ ,  $c_2 \sim \mathcal{N}(0, I)$ 
5    $\mathcal{H} = G_2(\alpha^{(K)} \cdot c_1^{(K)}, c_2)$ 
6   if  $S(\mathcal{H}) < 0$  then
7     return  $\mathcal{H}$ 
8   else if  $S(\mathcal{H}) < s_{min}$  then
9      $\alpha^* = \alpha^{(K)}$ ,  $c_2^* = c_2$ ,  $s_{min} = S(\mathcal{H})$ 
10  end
11 end
12  $\gamma^* = logit(\alpha^*)$ 
13 for  $n \in [n_{gd}]$  do
14    $\alpha^* = sigmoid(\gamma^*)$ 
15    $\alpha^* = \alpha^* / sum(\alpha^*)$ 
16    $\mathcal{H} = G_2(\alpha^* \cdot c_1, c_2^*)$ 
17   if  $S(\mathcal{H}) < 0$  then
18     return  $\mathcal{H}$ 
19    $\gamma^* = \gamma^* - \eta \nabla_{\gamma^*} S(\mathcal{H})$ 
20    $c_2^* = c_2^* - \eta \nabla_{c_2^*} S(\mathcal{H})$ 
21 end

```

Figure 8 shows the case of simplex sampling being unable to find \mathcal{H} even after setting $n_{sim} \approx 100k$. Combining the respective strengths of sampling and gradient-descent, both of which use feedback from coverage indicators, it is often sufficient to sample for 10–50 iterations, with the subsequent gradient search taking no more than a few 10s of iterations to find \mathcal{H} . The example in Figure 7 may contain a single branching condition, but the method can be extended to multiple independent *if-else* conditions, even when they are nested. A search function, in this case, returns a vector containing one minimiz-

able coverage indicator per independent branching condition. Due to independence, this vector can be collectively minimized by executing one search per coverage indicator in parallel.

Using SilGAN, we thus demonstrate an end-to-end pipeline for scenario specification, stimulus generation and test automation. In demonstrating test automation, we however show only a case of code coverage with boolean operations made differentiable using coverage indicators. While the principle of targeted, combined sampling and gradient based search of the latent space is vital, additional measures may be necessary to scale the technique for automatically testing code with non-differentiable intermediate operations. We leave the investigation of such measures for future work.

VI. RELATED WORK

Deep generative models for time series have garnered a substantial amount of attention in a variety of applications. Examples include generating synthetic medical data to ensure privacy of the patients [15], modelling financial time series to forecast asset returns [16] and music creation [17]. In contrast, domain adaptation of time series has received sparser attention. Previous work has employed domain adaptation to extract physiologically invariant features in a clinical setting [18] and the creation of a dataset invariant to specific characteristics of individual blast furnaces [19]. Closest to our work is [7] that seeks to generate in-vehicle time series using a weak form of domain adaptation. We significantly improve on that work by introducing a simpler way of using templates and using it to set explicit objectives for domain adaptation. This greatly reduces specification effort and eases generation. Moreover, we enrich test stimulus generation through multi-modal translation using disentangled information domains and expansion. The technique for targeted search further makes it much more conducive for SIL testing. While parallels to our overall approach are rare in the domain of time series, certain aspects have analogies in the image domain. Examples include sketch to image [20], [13], [12], and image outpainting [21].

Prior research on embedding methods for time series include [22] that trains general purpose embeddings through tokenization and skip-gram techniques. By exploiting the fact that similarity matrices can be of low rank, [23] converts time series into a matrix representation suitable for clustering techniques. In [24], a twin-autoencoding structure is employed to learn embeddings invariant to warped transformation. Our technique for embedding differ by disentangling time series at different levels of abstractions into components with distinct interpretations, apt for a testing pipeline.

Previous work on deep learning for test case generation include time series generation for automotive SIL testing [7], text generation for testing mobile apps [25], and protocol frame generation for testing process control equipment [26]. Parallel work on dynamic software testing, include identifying worst case branching for stress testing [27] and GUI testing for apps [28]. Unlike these methods which target specific aspects of testing in different domains, by introducing techniques for specification, generation, and test automation,

we demonstrate an end-to-end framework for SIL testing of vehicle control software.

Finally, while we show a simple example of using the trained model for SIL test automation, in extending our work to more general cases, previous work on ways to smooth boolean conditions [29] [30] may be helpful.

VII. CONCLUSIONS

The future of automotive software testing depends upon techniques that accelerate feedback without compromising quality. Exploiting the ready availability of data from field vehicles and advances in deep learning, we present a domain translation model that helps achieve both. Using this model, we demonstrate how to specify and generate realistic driving maneuvers at a granular level, easing the test of low-level vehicle control software. Further, we also introduce techniques of targeted generation that can automate test objectives like code coverage. While improvements such as scaling the automation procedure to deal with other test objectives can strengthen the method, its fundamental principles significantly increase the credibility of simulation-driven testing as a quicker and viable alternative to field tests.

VIII. ACKNOWLEDGEMENTS

We thank Carl Seger and Henrik Lönn for helpful discussions. This work is supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

- [1] A. Haghighatkhan, A. Banijamali, O. Pakanen, M. Oivo, and P. Kuvaja, “Automotive software engineering: A systematic mapping study,” *J. Syst. Softw.*, vol. 128, pp. 25–55, 2017.
- [2] J. Stolfa, S. Stolfa, C. Baio, U. Madaleno, P. Dolejsi, F. Brugnoli, and R. Messnarz, “DRIVES - EU blueprint project for the automotive sector - A literature review of drivers of change in automotive industry,” *J. Softw. Evol. Process.*, vol. 32, no. 3, 2020.
- [3] H. Kaijser, H. Lönn, P. Thorngren, J. Ekberg, M. Hemmingsson, and M. Larsson, “Towards Simulation-Based Verification for Continuous Integration and Delivery,” in *ERTS 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), Jan. 2018.
- [4] D. Nalic, T. Mihajl, M. Bäumler, M. Lehmann, and S. Bernsteiner, “Scenario based testing of automated driving systems: A literature survey,” p. 1, 11 2020. FISITA Web Congress 2020.
- [5] S. Ulbrich, T. Menzel, A. Reschka, F. Schuld, and M. Maurer, “Defining and substantiating the terms scene, situation, and scenario for automated driving,” in *IEEE 18th International Conference on Intelligent Transportation Systems, Gran Canaria, Spain, September 15-18*, IEEE, 2015.
- [6] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta, “Formal scenario-based testing of autonomous vehicles: From simulation to the real world,” in *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23*, pp. 1–8, IEEE, 2020.
- [7] D. Parthasarathy, K. Bäckström, J. Henriksson, and S. Einarsdóttir, “Controlled time series generation for automotive software-in-the-loop testing using gans,” in *IEEE International Conference On Artificial Intelligence Testing, AITest 2020, Oxford, UK, August 3-6*, IEEE, 2020.
- [8] J. Greenyer, M. Haase, J. Marhenke, and R. Bellmer, “Evaluating a formal scenario-based method for the requirements analysis in automotive software engineering,” *ESEC/FSE 2015*, (New York, NY, USA), p. 1002–1005, Association for Computing Machinery, 2015.
- [9] G. Wilson and D. J. Cook, “A survey of unsupervised deep domain adaptation,” *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 5, 2020.
- [10] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, “Least squares generative adversarial networks,” in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 2813–2821, IEEE Computer Society, 2017.
- [11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, vol. 27, pp. 2672–2680, Curran Associates, Inc., 2014.
- [12] X. Huang, M. Liu, S. J. Belongie, and J. Kautz, “Multimodal unsupervised image-to-image translation,” in *15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part III*, vol. 11207 of *Lecture Notes in Computer Science*, pp. 179–196, Springer, 2018.
- [13] J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 2242–2251, IEEE Computer Society, 2017.
- [14] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.
- [15] C. Esteban, S. L. Hyland, and G. Rätsch, “Real-valued (medical) time series generation with recurrent conditional gans,” *CoRR*, vol. abs/1706.02633, 2017.
- [16] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, “Quant gans: Deep generation of financial time series,” *CoRR*, vol. abs/1907.06673, 2019.
- [17] O. Mogren, “C-RNN-GAN: continuous recurrent neural networks with adversarial training,” *CoRR*, vol. abs/1611.09904, 2016.
- [18] P. Gupta, P. Malhotra, J. Narwariya, L. Vig, and G. Shroff, “Transfer learning for clinical time series analysis using deep neural networks,” *CoRR*, vol. abs/1904.00655, 2019.
- [19] C. Schockaert and H. Hoyez, “Mts-cyclegan: An adversarial-based deep mapping learning network for multivariate time series domain adaptation applied to the ironmaking industry,” *CoRR*, vol. abs/2007.07518, 2020.
- [20] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 5967–5976, IEEE Computer Society, 2017.
- [21] M. Sabini and G. Rusak, “Painting outside the box: Image outpainting with gans,” *CoRR*, vol. abs/1808.08483, 2018.
- [22] C. Nalmpantis and D. Vrakas, “Signal2vec: Time series embedding representation,” in *Engineering Applications of Neural Networks - 20th International Conference, EANN 2019, Heronissos, Crete, Greece, May 24-26, 2019. Proceedings*, vol. 1000 of *Communications in Computer and Information Science*, pp. 80–90, Springer, 2019.
- [23] Q. Lei, J. Yi, R. Vaculín, L. Wu, and I. S. Dhillon, “Similarity preserving representation learning for time series clustering,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 2845–2851.
- [24] A. Mathew, D. P, and S. Bhadra, “Warping resilient time series embeddings,” *CoRR*, vol. abs/1906.05205, 2019.
- [25] P. Liu, X. Zhang, M. Pistoia, Y. Zheng, M. Marques, and L. Zeng, “Automatic text input generation for mobile testing,” in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, IEEE / ACM, 2017.
- [26] H. Zhao, Z. Li, H. Wei, J. Shi, and Y. Huang, “Seqfuzzer: An industrial protocol fuzzing framework from a deep learning perspective,” in *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*, pp. 59–67, IEEE, 2019.
- [27] J. Koo, C. Saumya, M. Kulkarni, and S. Bagchi, “Pyse: Automatic worst-case test generation by reinforcement learning,” in *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*, pp. 136–147, IEEE, 2019.
- [28] M. Pan, A. Huang, G. Wang, T. Zhang, and X. Li, “Reinforcement learning based curiosity-driven testing of android applications,” in *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, ACM, 2020.
- [29] J. P. Inala, S. Gao, S. Kong, and A. Solar-Lezama, “REAS: combining numerical optimization with SAT solving,” *CoRR*, vol. abs/1802.04408, 2018.
- [30] J. L. Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi, and K. Åkesson, “Enhancing temporal logic falsification with specification transformation and valued booleans,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 12, pp. 5247–5260, 2020.