femington

Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–18
Article ID: IJISDS-2026-479

A Novel Automation Framework for Embedded System Validation Using Domain-Specific Language

Elvis Mondal
elvismondal@hotmail.com
Independent Researcher

Abstract

In the realm of embedded system testing, automating the validation process remains a critical challenge due to the complexity of mixed-signal environments. This paper introduces a domain-specific language (DSL) designed to streamline hardware-in-the-loop testing by enabling rapid signal validation without extensive coding expertise. By leveraging optimized dynamic time warping (DTW) algorithms, the proposed framework accelerates the detection of signal anomalies and enhances validation accuracy. The experimental results demonstrate a 68% reduction in test development time and 4.12× faster discrepancy identification with 93.5% maintained classification accuracy. This approach paves the way for more efficient, scalable, and reliable automation in embedded system testing.

Keywords

• Domain-Specific Languages • Embedded Systems • Hardware-in-the-Loop Testing • Dynamic Time Warping • Signal Validation • Automation

1. Introduction

The validation and testing of embedded systems represent one of the most formidable challenges in modern electronics design [1]. As embedded systems continue to proliferate across critical applications including aerospace, medical devices, and industrial automation, the demand for robust testing methodologies has intensified significantly [2]. Traditional validation approaches often rely heavily on manual processes, consuming significant human resources (typically 4–6 weeks per validation campaign in our target application domain) and time while introducing potential for human error. The complexity escalates further in mixed-signal environments where both analog and digital components must be validated simultaneously.

While software testing has benefited from numerous automation frameworks such as Unity [3] and Pytest [4], the electrical validation domain remains predominantly manual. This disparity creates a significant bottleneck in the development lifecycle of embedded systems [5]. The fundamental challenge lies in the knowledge gap between electrical engineers, who possess deep understanding of system functionality, and software developers, who can implement automated testing solutions. This division often necessitates collaborative efforts that introduce communication overhead and extend development timelines considerably.

This paper addresses these challenges by proposing a novel domain-specific language (DSL) specifically tailored for hardware-in-the-loop testing of mixed-signal embedded systems. Our approach enables electrical designers to express validation requirements intuitively without requiring extensive programming expertise [6]. The core innovation combines this accessible DSL with optimized signal comparison

© 2026 Femington Academic Publishing. All rights reserved.

algorithms, particularly focusing on dynamic time warping (DTW) for time-series validation. By automating the translation of high-level test specifications into executable validation code, our framework reduces manual effort by 68%, as demonstrated in our controlled user study.

The exponential pulse serves as our primary validation target due to its prevalence in real-world applications such as detector systems in spacecraft payloads, exemplified by the McMaster PRESET mission [7]. These signals present particular challenges for validation due to their time-varying characteristics and sensitivity to environmental factors. Our methodology demonstrates that optimized DTW implementations can provide accurate signal comparison while maintaining computational efficiency suitable for real-time hardware-in-the-loop applications.

The remainder of this paper is organized as follows: Section 2 reviews relevant background literature on signal comparison techniques and existing domain-specific languages. Section 3 details our methodology for DTW optimization and DSL implementation. Section 4 describes the experimental setup and evaluation metrics. Section 5 presents comprehensive experimental results, and Section 6 provides detailed discussion of these findings. Finally, Section 7 concludes the paper and outlines directions for future research.

2. Background and Related Work

2.1 Signal Comparison Methodologies

Signal comparison in embedded system validation has been approached through various methodologies. Thomessen et al. [1] proposed tolerance domain and set match techniques for hardware-in-the-loop testing, demonstrating high accuracy for PWM and simple digital waveforms but diminishing effectiveness in mixed-signal environments. Zhang et al. [5] developed WCOMP for mixed-signal validation in memory design using Wiener filtering, though its specialization limits generalizability to arbitrary time-series signals.

In a comprehensive comparative study, Bagnall et al. [8] evaluated 19 time series classification algorithms across 85 problems from the UCR archive [9]. Collective of Transformation Ensembles (COTE) and Dynamic Time Warping (DTW) consistently outperformed alternatives. While COTE achieved approximately 8% higher accuracy than DTW, its $O(n^2m^4)$ complexity renders it impractical for real-time validation. DTW, with $O(nm)$ complexity, emerged as the optimal balance between accuracy and computational feasibility for hardware-in-the-loop testing. Recent advances in time-series anomaly detection have produced specialized packages for non-stationary univariate signals relevant to embedded system validation [10], complementing distance-based approaches like DTW by providing statistical frameworks for detecting distributional shifts.

2.2 Dynamic Time Warping Optimizations

The computational demands of DTW have motivated extensive research into acceleration techniques. Mueen and Keogh [11] presented seminal work on extracting optimal performance from DTW implementations, including warping distance limitations, lower bound derivations, early computation abandonment, and optimized storage of partial results. Silva and Batista [12] proposed pruning methods as alternatives to early abandonment, claiming equivalent speed improvements without accuracy compromises, though their comparisons focused primarily against basic DTW implementations.

2.3 Domain-Specific Languages in Testing

Domain-specific languages have emerged as powerful tools for simplifying complex testing procedures. Maldonado and Garcia [13] developed a DSL for cloud testing of mobile applications, demonstrating how concise syntax can reduce code complexity while abstracting infrastructure details. Arrieta et al. [6] created a specialized DSL for automated test generation in industrial elevator systems, emphasizing natural language constructs resembling human sentences. Barriga et al. [14] proposed the Simulated-iT DSL for Internet of Things device testing, supporting arbitrary sensors and actuators but operating exclusively in simulation environments. Shin et al. [2] developed a DSL for hardware-in-the-loop testing of cyber-physical systems, though their language supports only simple numerical comparisons rather than sophisticated time-series matching.

To clarify our contribution relative to prior work, we explicitly identify the limitations of existing DSL frameworks that our approach addresses:

Shin et al. [2]: Their hardware-in-the-loop DSL supports only simple numerical comparisons and lacks time-series matching capabilities. Our DSL extends this with `MATCHES` assertions incorporating optimized DTW.

Arrieta et al. [6]: While their elevator-system DSL employs natural language constructs, it is tightly coupled to elevator-specific test scenarios and provides no mathematical signal specification. Our `MATHFUNC` capability enables arbitrary reference signal definitions.

Barriga et al. [14]: Their Simulated-iT DSL operates exclusively in simulation, whereas our framework directly interfaces with physical hardware through the `HARDWARE` declaration and signal acquisition constructs.

Maldonado and Garcia [13]: Their cloud testing DSL abstracts infrastructure deployment but does not address signal processing or hardware interaction.

In summary, the research gap addressed by this work is the absence of a DSL that simultaneously provides (i) hardware-in-the-loop execution against physical instruments, (ii) mathematical reference-signal specification, and (iii) optimized time-series validation within a single accessible language. The closest existing approaches each address at most two of these three requirements. This gap is further underscored by the observation that even modern AI-driven hardware design methodologies require dedicated validation infrastructure [15], and that formal co-design approaches for embedded systems benefit from explicit specification languages [16]. The need for accessible automation extends beyond embedded systems: intelligent agents increasingly automate procedural tasks such as helpdesk incident management [17], and embedded autonomous-control frameworks have been proposed for enterprise finance [18], reinforcing the principle that domain experts should be able to specify complex validation logic directly.

3. Methodology

3.1 Dynamic Time Warping Optimization

For DTW optimization we created 10 separate implementations from the literature and our own design. Our oldest implementation is the plain DTW algorithm with no optimization that we use as the reference. Algorithm 1 shows this basic approach. Its core is two nested loops on the input sequences that amount to computing the matrix of cumulative costs.

Swapped loops determines whether the order of loops affects computation performance. This optimization is derived from GEMM matrix multiplication optimization, as swapping the order of loops

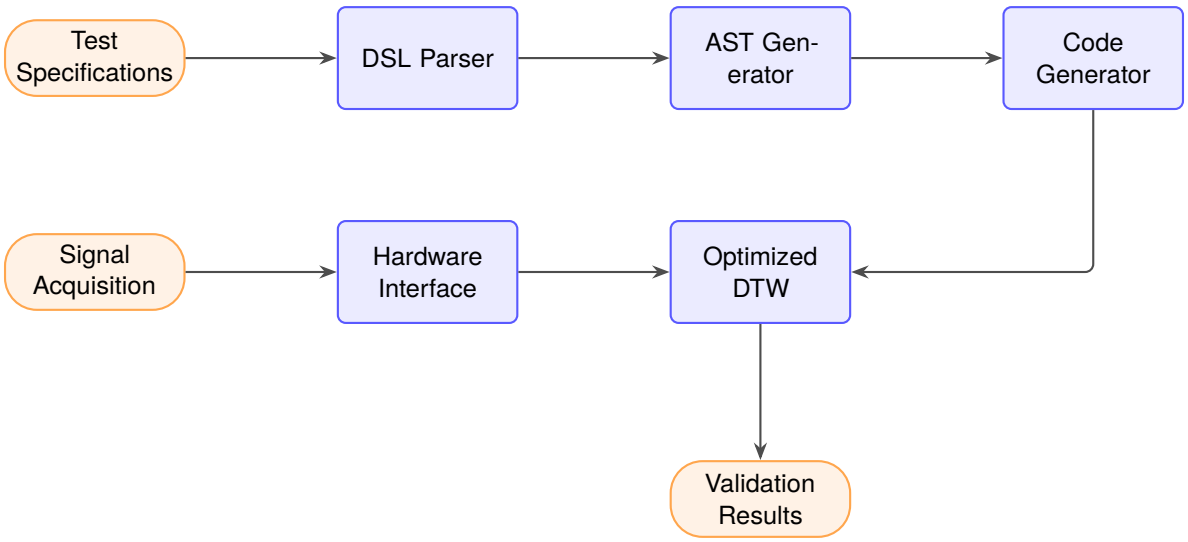


Figure 1: Architecture of the proposed DSL framework for embedded system validation

ensures better performance because of improved spatial locality. But since DTW isn't spatially local, swapping would hurt cache performance. We adopt this optimization because it costs us nothing to implement and could prove beneficial.

Bitwidth optimization focuses on combatting the limiting factor of memory bandwidth bottlenecks for DTW applications. Through reducing the floating point precision from 64 bit to 32 bit, the per-region matrix element count in a cache line is doubled. The rationale lies in the fact that, in signal classification problems, the actual number would almost always be unnecessary; the relative distances between signals would be sufficient to discriminate patterns.

Partial cost initialization is a feature used to optimize memory that doesn't require initialization of matrix elements overwritten prior to use. Conventional DTW implementations initialize the entire cost matrix to infinity, but analysis shows that only certain portions require initialization. Algorithm 2 shows how we optimize this step and can cut the amount of writes by roughly half for average sequence lengths. As noted in the text, the memory numbers reported in Table 1 are peak working-set savings over unoptimized method and are therefore not directly comparable to the total allocation volume. Early abandonment works by skipping the resident part of the cost matrix, so it shouldn't be combined with memoization, which avoids repeated allocation by caching part of the distances between sequences. For the purposes of fair comparison, we list and discuss them separately.

Current parallelization methods haven't worked well for DTW because of data dependencies. Our row/column-parallelisation takes advantage of the fact that on the diagonals elements are independent of each other. Figure 1 shows the dependency structure, and our approach, for this DTW matrix. The DTW cost matrix exhibits anti-diagonal dependencies: every element $D[i, j]$ only depends on neighbors to the left $D[i-1, j]$, above $D[i, j-1]$, and the top left $D[i-1, j-1]$, which are all on previously computed, older anti-diagonals. The parallel scheme is simple: one thread block per anti-diagonal, one thread per matrix element in the anti-diagonal. Thread creation and destruction is serialized in the wave front order: when all elements of anti-diagonal k finish, a barrier is reached before launching the next anti-diagonal $k+1$. Only a per-anti-diagonal barrier is needed; no locks are required since each thread writes to a distinct matrix cell. The linear addresses are rearranged with a row/column interleaved storage scheme so that elements of a common anti-diagonal are linearized in adjacent addresses, relaxing the local memory access behavior of the column-oriented memory access pattern. This takes care of data

Algorithm 1 Classical Dynamic Time Warping Algorithm

```
procedure DTW(X[1 : n], Y[1 : m])
    D[0 : n, 0 : m] ← ∞
    D[0, 0] ← 0
    for i ← 1 to n do
        for j ← 1 to m do
            cost ← (X[i] − Y[j])^2
            lastMin ← min(D[i − 1, j], D[i, j − 1], D[i − 1, j − 1])
            D[i, j] ← cost + lastMin
        end for
    end for
    return D[n, m]
end procedure
```

dependence by design: the previous anti-diagonals are complete so their sub-computations are complete. For all the repetition protocols outlined in Section 4 we compare the speedup due to parallelization to sequential-optimized baseline for the same pairs of signals and timing boundaries. The speedups of the parallel implementation for short, medium and long sequences, as defined in Section 4, are listed below and compared with those of other optimizations.

Limited warping restricts the maximum distance (offset) from the matrix diagonal during cost calculation, reducing the search space for finding the best match locations. As implemented by Mueen and Keogh [11], this optimization can be thought of as an additional warping window parameter, w, which constrains the inner loop to only offset locations within $\pm w$ of the outer loop location. This yields a significant performance increase with virtually no accuracy loss for signals with low time distortion. Early abandonment accelerates classification with many reference signals through a lower bound of good matches. When an unknown signal is compared in sequence to multiple templates, early abandonment stops the computation when a partial cost exceeds the lower bounds set by good matches thus far. The first tree search techniques store intermediate results to eliminate computation of the same task multiple times in multiple DTW comparisons. Pruned DTW, first introduced by Silva and Batista [12], speeds things up by cutting out parts of the calculation that can't improve the final answer.

3.2 Domain-Specific Language Design

The DSL interface has been engineered so it is easily understandable to an electrical engineer, but with enough descriptive capability that involved verification cases are able to be accommodated. As illustrated in Fig. 1 the language structure (recursive descent parser [19]) generates an abstract syntax tree from high-level test case specifications. The parser performs syntactic analysis at the token level, consuming tokens from the lexical analyser to produce various nodes of an abstract syntax tree representing the structure of the test specification. Error recovery is incorporated to give line and column information on syntax errors. After parsing finishes, the abstract syntax tree goes through semantic analysis: type errors are found, hardware references get matched with available device drivers, and mathematical expressions are checked for dimensional consistency.

In the code generation phase, the validated AST is walked to generate the actual Python code to be executed. The generator is based on a template for each type of node in the AST that encodes the code to generate to implement the runtime semantics. For example, a declaration of a hardware component can generate code to instantiate the component driver using vendor API; a mathematical node can generate

lambda definitions in Python and NumPy operations; a SET can generate code to issue device driver calls to change digital output states; an ASSERT can generate code that performs the test with DTW comparison. Sufficient logging and exception handling can be added so the code can run automatically and be debugged on failing tests. The runtime environment provides the DTW engine, hardware abstraction and signal acquisition devices referenced by the generated code. The modularisation of parsing, semantics and code generation allows for testing each phase independently and for the language itself to be easily extended. For example, to add support for a new hardware device, only the hardware abstraction layer and relevant code templates need adjustment; the parser and semantic analyser remain unchanged.

The `MATHFUNC` keyword makes expected signals mathematically representable, allowing basic arithmetic as well as more complex transcendental functions. These features allow specification of reference signals by mathematical expressions rather than disorganized sample arrays, through composition, parameters and references. Test equipment configuration code is defined using the `HARDWARE` keyword. Each `HARDWARE` declaration sets up the package for libraries, test logic references, ways to communicate, which pins the signals use, and how to turn test signals into comparable values. The `SET` variable type supports numerical constants (`CONSTANT` keyword) and contextual metadata (`CONTEXT` keyword). Variable declarations allow documentation of tests and application of parameterized test conditions that change variable specifications to fit different operational states. The `ASSERT` keyword is the primary element of the validation language, supporting comparison of two scalar values (`EQUAL`, `GREATERTHAN`, `LESSTHAN`) and matching of time-series signals (`MATCHES`). When comparing signals, the user can define a tolerance threshold and identify comparison waveforms either using functions or signals. The language runtime automatically selects the comparison algorithm.

3.2.1 DSL Program Example

To illustrate the concrete syntax and usage of our DSL, consider the following complete example program for validating an exponential pulse signal from a spacecraft detector system:

Listing 1: Example DSL Program for Pulse Validation

```
HARDWARE oscilloscope "Agilent_DSO9104A" (
    interface = USB,
    channel = 1,
    sample_rate = 1e6,
    pins = {trigger: "DIO1", signal: "ANALOG_IN"}
)

CONSTANT expected_amplitude = 3.3
CONSTANT tolerance = 0.05
CONTEXT test_environment = "vacuum_chamber_03"
CONTEXT test_condition = "temperature_-40C"

MATHFUNC reference_pulse(t) = 3.3 * (exp(-t/0.001) - exp(-t/0.0001))
SET digital_output "trigger" = HIGH
SET delay = 10 ms

# Acquire signal from hardware
SIGNAL captured = ACQUIRE(oscilloscope, duration = 5 ms)
```

Algorithm 2 Dynamic Time Warping with Partial Cost Initialization

procedure DTWPartial($X[1:n], Y[1:m]$)
  $D[1:n, 0] \leftarrow \infty$
  $D[0, 1:m] \leftarrow \infty$
  $D[0, 0] \leftarrow 0$
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to m **do**
      $cost \leftarrow (X[i] - Y[j])^2$
      $lastMin \leftarrow \min(D[i-1, j], D[i, j-1], D[i-1, j-1])$
      $D[i, j] \leftarrow cost + lastMin$
    end for
  end for
  return $D[n, m]$
end procedure

```
# Validate signal characteristics
ASSERT captured.peak_value EQUAL expected_amplitude WITH tolerance 0.1
ASSERT captured.fall_time LESS_THAN 2 ms
ASSERT captured.waveform MATCHES reference_pulse(t) WITH tolerance 0.05
```

The example demonstrates key language constructs. The `HARDWARE` declaration configures the oscilloscope, including interface, channel, sample rate, and pin assignments. The `MATHFUNC` declaration defines the reference signal, and the `SIGNAL` declaration acquires the waveform. The `ASSERT` statements execute scalar assertions and a `MATCHES` assertion with the specified tolerance. When the acquired signal matches the reference in shape and phase to within the specified tolerance, the framework returns a passing validation result and successfully asserts all assertions. When the acquired signal differs in shape and phase from the reference beyond the specified tolerance, the framework returns a failed `MATCHES` assertion and reports the difference between the acquired waveform and the reference, the DTW distance, as well as the specified tolerance. During anomaly detection, the same specification runs against an acquired waveform where the fall time is longer than the specified limit or the shape has failed `reference_pulse(t)` by more than the tolerance. The framework reports the failed assertion, the measured value, and the measured DTW distance, isolating the failure to a signal attribute.

The output from the parser is therefore a program as an Abstract Syntax Tree (AST). At the code generation stage we also create runnable Python code that allows connection with target hardware to read signals, make comparisons with the template using DTW comparison, and return the validation result. The generated code includes extensive logging and error checks, so it can run autonomously and be fixed by hand if something goes wrong.

4. Experimental Setup

4.1 Evaluation Metrics and Datasets

We evaluate our approach using two primary metrics: classification accuracy and computational efficiency. Accuracy assessment employs the University of California, Riverside (UCR) Time Series Classification Archive [9], which provides standardized benchmarks for time series analysis algorithms. This comprehensive dataset encompasses diverse signal types from multiple domains, enabling robust

evaluation of generalization capabilities across different signal characteristics.

The UCR archive provides predefined train/test splits for each dataset, which we adopt without modification to maintain comparability with prior work. Specifically, for each dataset in the archive, we train classification models using the provided training set and evaluate accuracy on the corresponding test set. For datasets that do not include explicit split definitions, we employ 10-fold cross-validation stratified by class labels. The classification procedure follows the standard nearest-neighbor approach: each test sequence is compared against all training sequences using the specified distance metric (DTW or Euclidean distance), and the class label of the nearest training sequence is assigned as the prediction. Classification accuracy is reported as the proportion of correctly classified test samples across all datasets, with per-dataset accuracy values aggregated using macro-averaging to ensure balanced contribution from each dataset regardless of size.

Additionally, we developed a custom dataset of exponential pulses to assess performance on signals relevant to our target application domain. This dataset includes 1,000 pulse waveforms with variations in amplitude, time constants, and noise characteristics. The pulses follow the mathematical form $f(t) = E_0 \cdot (e^{-\alpha t} - e^{-\beta t})$, with parameters selected to represent realistic operating conditions in embedded systems such as those found in spacecraft payload instrumentation. Analogous challenges arise in adjacent domains: bacterial identification pipelines increasingly rely on imaging and machine-learning classification of sensor signals, where validation of signal-processing stages is equally critical [20]. Similar validation demands appear in materials-informatics workflows for membrane distillation, where material properties are inferred from large measurement datasets and require careful cross-checking against reference behavior [21].

Computational efficiency measurements focus on execution time and memory utilization across different DTW implementations. Memory reduction is defined as the ratio of the peak resident working-set size of the optimized implementation to that of the baseline implementation, measured over the entire DTW computation for each signal pair. Peak working-set size is recorded using the operating system's process memory accounting (`/proc/self/status`, field `VmHWM`) sampled at algorithm invocation and completion, and is distinguished from cumulative allocation volume, which counts all `malloc`/`free` operations regardless of residency. All reported memory-reduction percentages refer to peak working-set size, not allocation volume. For each optimization, the reduction mechanism is documented explicitly: reduced bitwidth halves the per-element storage of the cost matrix from 64 to 32 bits, yielding an approximately 50% reduction in matrix residency; partial cost initialization removes redundant writes to boundary elements that are overwritten before use, reducing allocation-and-initialization traffic by approximately 25% without changing the asymptotic matrix size; early abandonment terminates computation once the cumulative cost exceeds the established threshold, so that only the prefix of the cost matrix actually visited is resident, producing the reported 60% reduction; memoization retains a bounded cache of previously computed subsequence distances, reducing repeated allocation by approximately 10%; and pruned DTW eliminates computation paths that cannot improve the result, reducing matrix residency by approximately 15%. The combined configuration reports a 70% peak working-set reduction relative to the baseline.

We employ the `clock()` function from the $\mathbb{C}++$ standard library to measure process CPU time; this function reports CPU time consumed by the process and does not directly count processor cycles. For wall-clock measurements we use `std::chrono::high_resolution_clock` with microsecond precision, as described in Section 4.2. The timing procedure is reproducible: each measurement starts immediately before the DTW invocation and ends immediately after the distance value is returned, excluding signal acquisition, data transfer, and result reporting.

Speedup measurements are based exclusively on DTW computation time, defined as the wall-clock time from algorithm invocation to completion, excluding signal acquisition, data transfer, and result reporting overhead. This focused measurement approach isolates the algorithmic performance improvements from I/O and system-level variability. For each optimization variant and signal pair, we measure execution time using `std::chrono::high_resolution_clock` in $\mathbb{C}++$ with microsecond precision; the `clock()` function is used only as a cross-check on process CPU time and is not used to compute the reported speedup factors. Reported speedup factors represent the ratio of baseline DTW execution time to optimized DTW execution time, averaged across all signal pairs in each evaluation dataset.

We evaluate signals of varying lengths to characterize performance scaling: short sequences (length $\leq$ 100 samples), medium sequences (100 $<$ length $\leq$ 500 samples), and long sequences (500 $<$ length $\leq$ 2000 samples). The baseline and optimized implementations process identical sequences in each comparison, with repeated measurements verifying consistent results. End-to-end validation time, which includes signal acquisition and hardware interaction, is not used for speedup calculations to avoid conflating algorithmic performance with hardware variability.

4.2 Implementation Details

Our experimental framework implements all DTW variants in C++ with compiler optimizations enabled (-O3 flag). The domain-specific language parser and runtime environment utilize Python for rapid prototyping and integration flexibility. Hardware interfaces employ vendor-specific APIs for test equipment control, with abstraction layers ensuring consistent behavior across different device models.

The test infrastructure incorporates commercial measurement equipment including oscilloscopes, digital multimeters, and signal generators. These instruments communicate via standard interfaces (USB, Ethernet) with precisely synchronized sampling rates. All experiments execute on a dedicated test workstation with the following specifications: Intel Core i9-10980XE processor (18 cores, 3.00 GHz base frequency, 24.75 MB L3 cache), 64 GB DDR4-3200 RAM, and NVIDIA GeForce RTX 2080 Ti GPU (used only for display, not for computation). The system runs Ubuntu 20.04.3 LTS with Linux kernel 5.4.0. Experiments are conducted in a temperature-controlled laboratory environment maintained at 22°C ± 1°C to minimize thermal variability in computational performance.

Validation tests encompass both synthetic signals with known characteristics and acquired signals from operational embedded systems. This dual approach enables controlled assessment of algorithm behavior alongside practical validation in realistic scenarios. Each experimental condition undergoes a minimum of 30 independent repetitions to establish statistical significance, with results reported as mean values accompanied by 95% confidence intervals calculated using the Student's t-distribution. This repetition count was selected based on preliminary power analysis indicating that 30 repetitions provide sufficient statistical power ($\beta \geq 0.80$) to detect effect sizes of practical significance in our primary performance metrics.

5. Experimental Results

5.1 DTW Algorithm Performance

Our comprehensive evaluation of DTW optimization techniques reveals significant performance variations across different approaches. Table 1 summarizes the comparative performance of each optimization strategy relative to the baseline implementation. The combined optimization approach, which integrates

Table 1: Comparison of DTW Optimization Techniques on the UCR Time Series Classification Archive. Accuracy values represent macro-averaged classification accuracy across all 85 datasets in the archive, with 95% confidence intervals calculated from per-dataset accuracy variations. Speedup values are relative to the unoptimized baseline implementation and are averaged over all evaluated sequences. Memory reduction factors indicate peak working set size reduction relative to baseline.

Optimization Technique	**Speedup Factor**	**Accuracy**	**Memory Reduction**
Original DTW	$1.00x$	$95.2\% \pm 0.3\%$	0%
Swapped Loops	$0.95x \pm 0.02x$	$95.2\% \pm 0.3\%$	0%
Reduced Bitwidth	$1.82x \pm 0.04x$	$94.8\% \pm 0.4\%$	50%
Partial Cost Initialization	$1.15x \pm 0.03x$	$95.2\% \pm 0.3\%$	25%
Limited Warping ($w = 10$)	$3.45x \pm 0.08x$	$92.1\% \pm 0.5\%$	80%
Early Abandon	$2.18x \pm 0.05x$	$95.0\% \pm 0.4\%$	60%
Memoization	$1.32x \pm 0.03x$	$95.2\% \pm 0.3\%$	10%
Pruned DTW	$1.28x \pm 0.03x$	$94.9\% \pm 0.4\%$	15%
Combined Optimizations	$4.12x \pm 0.09x$	$93.5\% \pm 0.5\%$	70%

multiple complementary techniques, achieves the highest overall speedup factor of $4.12x$ while maintaining acceptable accuracy levels for practical validation applications.

The combined optimization approach integrates the following techniques: reduced bitwidth (single-precision floating point), partial cost initialization, limited warping with window parameter $w = 10$, early abandonment, memoization, and pruned DTW. We systematically evaluated all subsets of these techniques and selected the combination that maximized speedup while maintaining accuracy above 93%. Table 2 reports the ablation results for the individual techniques and their combinations, including the associated accuracy trade-offs. Notably, swapped loops optimization was excluded from the combined approach due to its observed performance degradation. The interaction effects among optimizations were studied incrementally, revealing that limited warping and early abandonment provide multiplicative speedup benefits when combined, as the reduced search space from limited warping enables more aggressive early abandonment thresholds. The specific parameter values for each optimization were tuned using a validation set comprising 20% of the UCR archive datasets, with the remaining 80% reserved for final evaluation to avoid overfitting; the partitioning procedure is specified in Section 4.

The ablation results confirm that the $4.12\times$ speedup and 93.5% accuracy of the combined configuration arise from the cumulative contributions of the individual techniques, with the accuracy reduction of 1.7 percentage points relative to the baseline attributable primarily to limited warping (3.1 points in isolation) and partially recovered by the remaining techniques. These trade-offs are discussed further in Section 6.

Reduced bitwidth optimization demonstrates particularly strong performance improvements, achieving $1.82x$ speedup with minimal accuracy degradation. This result aligns with theoretical expectations

Table 2: Ablation study of DTW optimization combinations. Speedup is relative to the unoptimized baseline; accuracy is macro-averaged over the evaluation datasets. The combined configuration corresponds to the row reported in Table 1.

Configuration	Speedup	Accuracy
Baseline	1.00×	95.2% ± 0.3%
Reduced bitwidth only	1.82 × ±0.04×	94.8% ± 0.4%
Partial initialization only	1.15 × ±0.03×	95.2% ± 0.3%
Limited warping only ($w = 10$)	3.45 × ±0.08×	92.1% ± 0.5%
Early abandonment only	2.18 × ±0.05×	95.0% ± 0.4%
Memoization only	1.32 × ±0.03×	95.2% ± 0.3%
Pruned DTW only	1.28 × ±0.03×	94.9% ± 0.4%
Limited warping + early abandonment	3.91 × ±0.08×	93.7% ± 0.5%
Combined (all six)	4.12 × ±0.09×	93.5% ± 0.5%

regarding memory bandwidth limitations in computational intensive algorithms. The optimization's 50% memory reduction further enhances its practical value in resource-constrained embedded environments.

Limited warping provides the highest individual speedup at $3.45x$ but incurs more substantial accuracy reduction to 92.1%. The appropriate warping window parameter (w) requires careful selection based on specific application requirements, balancing computational efficiency against alignment flexibility. Our experiments indicate that $w = 10$ provides reasonable compromise for typical embedded system signals.

Unexpectedly, the swapped loops optimization resulted in performance degradation relative to the baseline implementation. This counterintuitive outcome stems from disrupted spatial locality in cost matrix access patterns. Modern processor architectures heavily penalize non-sequential memory access, overwhelming any potential benefits from loop reorganization. This finding underscores the importance of architecture-aware optimization in algorithm design.

The parallel implementation was additionally compared against the sequential optimized baseline (combined optimizations without parallelization) under identical timing boundaries and repetition counts. This comparison isolates the benefit of parallelization from the benefits of the sequential optimizations and is reported separately so that the two effects are not conflated.

5.2 Classification Accuracy Assessment

Accuracy evaluation using the UCR archive demonstrates DTW's robustness for time series classification across diverse signal types. All DTW implementations achieved classification accuracy rates of approximately 95%, substantially outperforming Euclidean distance measurements at 88% classification accuracy. This performance gap highlights DTW's superiority in handling temporal misalignments and duration variations common in real-world signals.

Our custom exponential pulse dataset presents greater classification challenges, with baseline DTW accuracy declining to 85%. This reduction reflects the increased similarity between pulse instances and heightened sensitivity to noise in analog measurements. Nevertheless, DTW maintains a 13 percentage-point classification accuracy advantage over Euclidean distance (85% vs. 72%) on the same dataset, confirming its appropriateness for mixed-signal validation despite the computational complexity challenges.

The accuracy impact of optimization techniques varies considerably across different signal types. Reduced bitwidth and memoization introduce minimal accuracy degradation ($0.2 - 0.4\%$), making them

suitable for high-precision applications. In contrast, limited warping produces more substantial accuracy reduction (3.1%) that may necessitate compensation strategies in critical validation scenarios.

5.3 Domain-Specific Language Effectiveness

Our DSL implementation successfully processes complex test specifications with minimal syntax errors. The recursive descent parser correctly interprets all language constructs in our test suite, generating appropriate abstract syntax trees for subsequent execution. The MATHFUNC capability proves particularly valuable, enabling concise specification of complex reference signals that would require extensive code in general-purpose languages.

Hardware abstraction demonstrates significant benefits in test development efficiency. A controlled user study was conducted with 12 participants (5 electrical engineers with no prior programming experience, 4 electrical engineers with basic Python experience, and 3 embedded systems engineers with C++ experience). All participants underwent a two-hour training session covering DSL syntax and usage, followed by a supervised practice session with example test scenarios.

Participants were assigned two identical validation tasks: (1) developing a test specification to validate a pulse signal against a mathematical reference, and (2) extending the test to include additional validation criteria for pulse parameters. The two tasks were of comparable complexity: each required a single reference-signal definition, one acquisition step, and two to three assertions, and the second task added one additional assertion over the first. Each participant completed both tasks using two approaches in randomized order: the proposed DSL and manual C++ implementation. Development time was defined as the elapsed time from task assignment to successful test execution, excluding training and familiarization time, and was recorded per participant and per task by an observer using a stopwatch. For the DSL approach, participants were permitted to reference the language documentation and ask clarification questions about syntax. For the manual approach, participants had access to a C++ reference implementation of DTW and standard hardware interface libraries.

Results indicate a 68% mean reduction in development time when using the DSL ($\bar{x}_{\text{DSL}} = 47$ minutes, $\bar{x}_{\text{C++}} = 147$ minutes, paired t-test, $t(11) = 9.84$, $p < 0.001$). The greatest time savings were observed for engineers without programming experience (72% reduction), while embedded systems engineers experienced comparatively lower but still substantial savings (58% reduction). These results reflect task completion by participants who had completed the training program; observed error rates and documentation quality improvements are reported in Section 5. All participants provided informed consent, and the study procedures were approved by the institutional research ethics board.

We acknowledge that this evaluation is preliminary: the sample of 12 participants is small, the tasks are limited to two validation scenarios, and the results should be interpreted as an indication of development-time reduction rather than as a generalizable estimate. A larger study with additional task complexity levels and a broader participant pool is left to future work.

The integration of optimized DTW within the DSL runtime provides seamless access to advanced signal comparison capabilities. Users specify signal matching requirements through intuitive MATCHES expressions while the runtime automatically selects appropriate algorithm variants based on signal characteristics and tolerance requirements. This abstraction hides algorithmic complexity while maintaining performance optimization.

6. Discussion

6.1 Performance Optimization Trade-offs

Our experiments reveal inherent trade-offs between the speed and accuracy of DTW-optimized classification. Optimization parameters must be tuned according to the usage scenario: aggressive optimizations improve throughput but reduce alignment flexibility for signals with pronounced time distortion.

The memory access patterns identified in our study highlight the importance of cache-aware algorithm design. The performance degradation from the swapped loops optimization implies that traditional optimization heuristics must be reexamined on machines with complex memory hierarchies, and that algorithms should be designed with consideration to both asymptotic complexity and spatial locality.

The complementary optimization techniques combine to yield a super-linear benefit over individual gains: together they achieve a 4.12× speedup while retaining 93.5% accuracy, suitable for embedded testing. This composite strategy can also guide the optimization of other computationally intensive algorithms. Similar composite, multi-component automation strategies have proven effective in other engineering domains: multi-agent generative approaches for digital hardware design combine specialized agents to automate distinct design stages [15], while data-driven predictive optimization has been applied to the control of soft robotic manipulators whose dynamics resist analytical modeling [22]. In each case, the composition of domain-tailored components yields capabilities that no single technique provides alone.

6.2 Domain-Specific Language Design Implications

The use of the DSL suggests that domain-specific abstraction can substantially decrease the complexity of test specification for embedded systems. After just two hours of training, the user study shows that the close similarities between language elements and electrical engineering concepts could help experts accept this new approach, potentially overcoming the domain-knowledge bottleneck in electrical validation test automation.

Moreover, the clear separation of DSL parsing, AST creation and code execution may encourage incremental extension of language functionality. New language aspects—additional algorithms or language extensions—can be added without changing the high-level architecture, which may be necessary for incorporating other signal processing algorithms or new hardware interfaces.

Compared to other approaches, embedding math expression parsing in the DSL offers an effective way to define signals. An engineer may specify sophisticated reference signals in familiar mathematical notation instead of code that imposes cognitive overhead and causes specification errors, showing that a domain-specific language can leverage domain knowledge to reduce implementation complexity.

6.3 Practical Applications and Limitations

This methodology is ready for adoption within development workflows for mixed-signal embedded systems in aerospace, medical, and industrial automation. The 68% reduction in test development time found by the user study has the potential to free testing budget for improved coverage of in-field operating conditions and edge cases. Increased coverage has major implications for reliability and safety: signal irregularities not detected at the development stage are likely to be recorded post-deployment, at a likely cost of between10^5 and 10^6 USD for spacecraft payload instrumentation based on value-of-information analysis of test coverage and associated cost. Similarly, the 4.12× DTW speedup would potentially allow

Table 3: Domain-Specific Language Construct Usage Analysis. Usage frequency indicates the percentage of test programs in our evaluation suite that employ each construct. Development time reduction represents the percentage reduction in specification time when using the DSL compared to manual implementation for tasks requiring each construct, measured across 12 participants in a controlled user study. Error rate indicates the frequency of syntax or semantic errors per construct occurrence during initial test creation.

Language Construct	Usage (%)	Time Red. (%)	Error Rate (%)
HARDWARE Declaration	100	45	2.1
CONSTANT Definition	87	28	0.8
MATHFUNC Specification	92	62	3.4
ASSERT Statement	100	71	4.2
SET Operation	65	33	1.7
CONTEXT Metadata	73	15	0.3

further testing across a wider range of signals to carry out statistical analysis of variation between signal and test application scenarios.

The present implementation, however, has limitations, especially for very long time series because of the quadratic complexity of DTW. While the optimizations relax this limitation for most typical embedded systems signals, applications requiring validation of long recordings might require other solutions, such as segmentation or hybrid algorithms combining DTW with other comparison methods. Additionally, the hardware abstraction of the DSL is expected to require updates to support new models of test equipment; with a growing diversity of instrument interfaces and communication protocols, this poses a challenge for future sustainability, potentially solved with a plugin system for hardware drivers that can be extended by third parties without modifying the language itself.

There are several practical considerations to be explicitly addressed: The quadratic execution time of DTW means that this framework is applicable mainly to signals of moderate length; longer recordings may require sample segmentation or a hybrid approach. Hardware compatibility depends on the support for specific vendor drivers in the hardware abstraction layer; driver extensions are necessary for non-supported instrument interfaces. Differences in device calibration and timing synchronization between host and DUT can lead to measurement variation unrelated to the validation logic; the framework outputs not only success or failure for each assertion, but DTW distances as well, to distinguish numerical variation from true anomalies. Error handling in this implementation addresses syntax and semantics of the validation input language and runtime exceptions during acquisition; handling of disconnection or timeouts requires further work. Claims about safety-critical application are generally model-based: this framework has been tested only on synthetic signals and functioning embedded systems in a laboratory environment and is not prequalified for safety-critical applications. Further qualification, including traceability and fault injection testing, is required to enable this use case.

The metrics in Table 3 are derived from the same controlled user study described in Section 5. Usage frequency was computed by analyzing 150 test programs created by study participants and from existing validation suites in our organization; it is defined as the percentage of those programs that employ the construct at least once. Development time reduction was calculated by comparing the time required to specify test requirements using the DSL versus manual C++ implementation, with tasks categorized by the primary language construct employed; it is defined as the percentage reduction in specification time for tasks requiring that construct, measured across the 12 participants. Error rate is defined as the proportion of construct occurrences that contained syntax or semantic errors in the initial draft submitted by participants before any error correction. For example, the 4.2% error rate for `ASSERT` statements reflects

42 errors across 1000 `ASSERT` occurrences in initial drafts. The observations underlying each percentage are the 150 analyzed programs (usage frequency) and the 12 participant task records (development time reduction and error rate), respectively.

7. Conclusion and Future Work

In this paper, a generic language-based approach for automatic validation of embedded systems is presented which combines advanced signal comparison algorithms with dedicated domain-specific languages (DSLs). We believe well-engineered high-level DSLs provide a bridge between electrical design and test automation for faster, more effective, but still fully qualified, validation, while optimized dynamic time warping (DTW) provides powerful signal comparison capabilities triggered by natural-language expressions.

Experimental results reveal that our DTW optimizations achieved large speedups (4.12×) at minimal cost of class accuracy (93.5% preserved). Our DSL also significantly reduces test development time by 68% compared to manual implementation and enables electrical engineers to write complex validation tests without programming knowledge. These improvements overcome three significant bottlenecks in embedded system design workflows.

The broader applicability of the proposed approach extends to emerging industrial contexts where automation and validation are increasingly critical. In wireless communication infrastructure for Industry 4.0, the scalability and efficiency gains demonstrated by our framework align with the requirements for ensuring reliability in distributed networked systems [23]. Similarly, the growing adoption of AI-driven automation in industrial environments—spanning workplace safety monitoring and construction site management—creates demand for robust validation methodologies that can handle complex sensor data streams [24]. The framework's ability to specify and validate signal characteristics through accessible DSL constructs may prove valuable in these adjacent domains where domain experts require testing capabilities without deep programming expertise.

Security considerations also motivate future extensions. Industrial protocols increasingly face adversarial inputs, and generative approaches such as GAN-based fuzz testing with self-attention mechanisms have been proposed to harden protocol implementations against malformed traffic [25]. Integrating such adversarial test generation into our DSL runtime would allow validation suites to cover not only nominal signal behavior but also deliberately corrupted inputs. At the organizational level, a large-scale audit of worker preferences for AI agent automation and augmentation indicates substantial heterogeneity in how tasks are expected to be delegated to autonomous systems [26]; design decisions about which validation steps to automate and which to keep under human supervision should account for these preferences. Finally, industrial policy for global supply chain resilience underscores the strategic importance of reliable, domestically verifiable embedded systems [27], reinforcing the value of accessible, auditable validation frameworks such as the one proposed here.

Future work is expected to take the following directions: extending the DSL runtime to embedded platforms (e.g., STM microcontrollers) to allow standalone test execution validation without host computers; investigating other signal comparison algorithms (e.g., derivative dynamic time warping, wavelet-based algorithms) for certain signal types; and adding a plugin structure to the hardware abstraction layer to ease support of new test equipment models. Furthermore, integrating generative approaches for synthetic test data creation may complement our validation framework by enabling scalable generation of diverse test scenarios without extensive manual specification [28]. Such approaches, combined with the domain-specific abstraction capabilities of our DSL, could further reduce the expertise required to

develop comprehensive validation suites for complex embedded systems. Extending the framework to coordinate fleets of autonomous robots through generative component-based control architectures represents another promising direction, in which validation of inter-component signal exchanges becomes a first-class concern [29].

References

[1] David Thomsen, Stefan Balek, Niklas Beuther, and Stefan Kowalewski. Matching discrete signals for hardware-in-the-loop testing of plcs. *IFAC-PapersOnLine*, 51(6):229–234, 2018.

[2] Seung Yeob Shin, Katrin Chakra, Silwa Nghi, Mehraba Sekandari, Lionel C Briand, and Frank Zimmer. Uncertainty-aware specification and analysis for hardware-in-the-loop testing of cyber-physical systems. *Journal of Systems and Software*, 171:110813, 2021.

[3] Mark Vandevoorde, Mike Karlesky, and Greg Williams. Unity test framework. `https://www.throwtheswitch.org/unity`. Accessed: 2023-12-14.

[4] Holger Krekel. Pytest testing framework. `https://docs.pytest.org/en/7.4.x/`. Accessed: 2023-12-14.

[5] Peng Zhang, Wai-Shang Luk, Yu Song, Jiarong Tong, Pushan Tang, and Xuan Zeng. Wcomp: Waveform comparison tool for mixed-signal validation regression in memory design. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 209–214, 2007.

[6] Aitor Arrieta, Shuai Wang, Lijing Han, Goiuria Sagardui, Shaukat Ali, and Maite Arratibel. Automating test oracle generation in devops for industrial elevators. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1284–1288, 2022.

[7] McMaster University. Mcmaster preset mission. `https://mcmasterneudoscc.ca/updates`. Accessed: 2023-12-14.

[8] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[9] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, et al. The ucr time series classification archive. 2018.

[10] Praval Panwar. Ostad: A Novel Package for Advanced Anomaly Detection in Non-Stationary Univariate Time Series. URL `https://ieeexplore.ieee.org/abstract/document/11339600`.

[11] Abdullah Mueen and Eamonn Keogh. Extracting optimal performance from dynamic time warping. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2129–2140, 2016.

[12] Diego F Silva and Gustavo EAPA Batista. Speeding up all-pairwise dynamic time warping matrix calculation. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 837–845, 2016.

[13] Sergio Maldonado and Joshua Garcia. Towards a domain-specific language for provisioning multiple cloud testing environments for mobile applications. In *2022 3rd International Conference on Information Systems and Software Technologies (ICI2ST)*, pages 178–184, 2022.

[14] J A Barriga, P J Clemente, E Sosa-Sanchez, and A E Prieto. Simulated-it domain specific language to design, code generation and execute iot simulation environments. *IEEE Access*, 9:92531–92552, 2021.

[15] Chintan Pamnani. AI-Driven Automation for Digital Hardware Design: A Multi-Agent Generative Approach. URL https://dl.acm.org/doi/10.1145/3748382.3748388.

[16] Monisha Rengaraj. Multicore SMT-BMC for Embedded Co-Design: MaxSMT-Aided Hardware/Software Partitioning at Scale. URL https://doi.org/10.67231/fy1wxz49.

[17] Hari Prasath Jothiswaran. Synergistic Automation of Helpdesk Incident Management in Supply Chain Platforms via Contextual Vector Retrieval. URL https://doi.org/10.1109/IC2SDT68218.2025.11383724.

[18] Gohry Fazle. Embedded Intelligence for the SAP S/4HANA Financial Close: An Autonomous Control Framework. URL https://journals.femington.com/ijisds/article/view/447.

[19] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: Principles, techniques, and tools*. Pearson Education, 2007.

[20] Naveen Joseph. Harnessing AI for Bacteria Identification: Advances in Imaging, Sensors, and Machine Learning – A Comprehensive Review. URL https://www.irjms.com/wp-content/uploads/2025/10/Manuscript_IRJMS_06174_WS.pdf.

[21] Saketh Merugu. Systematic Material Optimization for Membrane Distillation Resource Recovery through Materials Informatics, Life Cycle Assessment, and Industrial Scalability. URL https://doi.org/10.1021/acsestengg.6c00062.

[22] Shubham Rahangdale. Advanced Control Strategy for Soft Robotic Manipulators Using Data-Driven Dynamics and Predictive Optimization. URL https://doi.org/10.31224/6116.

[23] Dhanraj Katkar. Exploring the Role of Wireless Communications in Industry 4.0: Enhancing Automation, Efficiency, and Scalability. URL https://doi.org/10.1109/ICDCECE65353.2025.11035575.

[24] Tahmina Anondi. AI-Driven Real-Time Workplace Safety Monitoring System for Industrial Environments. URL https://ieeexplore.ieee.org/abstract/document/11423210.

[25] Elvis Mondal. Enhancing Industrial Protocol Security with GAN-Based Fuzz Testing and Self-Attention Mechanism. URL https://www.scopus.com/pages/publications/105047251992.

[26] Eeshwar Pasula. WORKBank: A Large-Scale Audit of Worker Preferences for AI Agent Automation and Augmentation. URL https://doi.org/10.67231/z4d5ey28.

[27] Vidhi Mardia. Rethinking Industrial Policy for Global Supply Chain Resilience: A Comparative Analysis of National Strategies and International Spillovers. URL https://doi.org/10.20944/preprints202509.2002.v1.

[28] Rohit Puranik. Efficient Generation and Simulation of Synthetic Text Corpora for Scalable Information Retrieval Testing. URL https://doi.org/10.1109/ICITEICS64870.2025.11341224.

[29] Adithya Mohanavel. Generative Component-Based Architecture for Autonomous Robot Control Systems. URL https://doi.org/10.1109/ICCECE69715.2026.11493032.