Automated Evolutionary Hyperparameter Tuning for NLP-Based Test Case Generation

Ivan Malashin, Igor Masich, Sergei Kurashkin, Andrei Gantimurov,
Aleksei Borodulin, Vladimir Neluyb, Vadim Tynchenko
Artificial Intelligence Technology Scientific and Education Center,
Bauman Moscow State Technical University
105005 Moscow, Russia
Emails: {imalashin, imasich, skurashkin, agantimurov, alexey.borodulin, vladimir.neluyb, vtynchenko}@emtc.ru

***Abstract*—Automated generation of executable test suites from natural-language requirements remains challenging due to linguistic ambiguity and sensitivity of generative models to decoding and training hyperparameters. This paper introduces a hierarchical, multi-level evolutionary framework that treats model hyperparameters and decoding strategies as upper-level decision variables and employs lower-level fitnesses that directly measure test-quality objectives (structural coverage, semantic diversity, redundancy, and runtime efficiency). The approach integrates retrieval-augmented grounding, surrogate-assisted preselection, lightweight LoRA adaptation and optional HIL evaluation. Empirical evaluation on PURE, PROMISE_exp and FR_NFR benchmarks (repeated runs, $n = 10$; paired two-sided t-tests, $\alpha = 0.05$) shows consistent gains: on PURE mean code coverage reaches 82.4% (vs. 75.1% for Bayesian optimisation and 68.9% for random search) with 145 unique scenarios and modest runtime overhead (≈58.3 s, ≈6% above Bayesian). Ablations confirm component effects (e.g., removing diversity reduces unique scenarios ≈18%; disabling the surrogate increases wall-clock ≈42%; disabling RAG drops grounded consistency ≈12%). Results indicate that co-optimising hyperparameters for explicit test-quality metrics, together with grounding and realistic execution, yields more useful, executable test suites. Future work will explore adaptive objective weighting, transfer warm-starts and probabilistic surrogates.**

***Index Terms*—automated test generation, hyperparameter optimization, evolutionary algorithms**

I. Introduction

Automating the generation of high-quality test cases from natural-language requirements remains a core challenge. Systems grow in complexity while requirements stay informal and ambiguous; manual test authoring is costly and hard to scale, and automated methods must reconcile linguistic uncertainty with the strict, executable semantics required by test oracles and execution environments.

Transformer-based generative models can synthesise diverse scenarios and executable scripts, but output quality is highly sensitive to configuration and decoding (learning rate, temperature, top-k/top-p, etc.). Optimising standard ML objectives (e.g., perplexity) does not reliably improve test-suite properties such as code coverage, semantic diversity, redundancy, or practical executability in hardware-in-the-loop (HIL) settings.

A hierarchical optimisation framework is proposed that links model hyperparameter selection to measurable test-quality objectives. An upper-level evolutionary search selects hyperparameters and decoding strategies, while a lower-level evaluator scores generated suites on structural and semantic metrics (coverage, diversity, redundancy, runtime). The decomposition keeps search tractable and admits domain-specific evaluators (RAG-grounded checks, HIL traces, static instrumentation).

The main contributions are:

- A multi-level evolutionary architecture that treats hyperparameters and decoding policies as upper-level decisions and uses test-quality metrics as lower-level fitnesses.
- Composite fitness metrics combining structural coverage, semantic diversity, redundancy reduction, and runtime constraints.
- A modular evaluation pipeline supporting retrieval grounding, HIL execution, and automated or expert-in-the-loop validation.
- Empirical evidence that the multi-level approach yields higher coverage and less duplication than baseline tuning methods.

The remainder of the paper is organised as follows. Section II reviews related work; Section III formalises the optimisation and metrics and describes the search algorithm and pipeline; Section V reports experiments; Section VI discusses limitations and future work; Section VII concludes.

II. Related Work

A. NLP for requirements understanding and test-case extraction

Automated extraction of test-relevant information from natural-language requirements underpins test generation. Template and boilerplate methods (Lim, 2024) parse actors, triggers and responses but struggle with negatives and ambiguity [1]. Graph and hybrid representations

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00051

(fuzzy graphs; hybrid NER+transformer) improve intermediate structure and entity normalization, stabilizing downstream generation [2], [3]. Annotation-projection and translation with expert validation enable cross-lingual bootstrapping [4], while end-to-end pipelines for Chinese IMT specs show comparable coverage to manual tests with less human effort [5].

B. LLMs and automated test-case generation

LLMs are effective at producing varied scenarios but have limits for strict extraction and labeling; discriminative models remain competitive for non-generation tasks [6], [7]. Hybrid pipelines combining LLMs with classical NLP and DL yield richer scenarios and deeper coverage in case studies [8], and retrieval grounding (RAG) improves context awareness and executability in generated code/tests [9]. In safety-critical settings, NLP-derived tests have been operationalized with HIL execution [10].

C. Optimization of generation systems and hyperparameter tuning

Generation quality is sensitive to hyperparameters (temperature, sampling, learning rate). Design-of-experiments and Taguchi approaches help tune GA and generation settings [11], while hierarchical and decomposition strategies for MINLP motivate multi-level optimisation of discrete/continuous decisions and evaluators [12], [13]. Efficiency and adversarial failure modes (e.g., EOS-delay attacks) must be considered when optimising production pipelines [14].

D. Test-quality metrics, assembly strategies and metamorphic testing

Beyond structural coverage, test quality requires measures of semantic diversity, redundancy and oracle presence. Controlled syntactic assembly and metamorphic variants produce effective test suites without explicit oracles [15]. Convergent strategies—vector retrieval, RAG, hybrid NER and expert annotation projection—increase semantic coverage while reducing irrelevant outputs [3], [4], [9]; empirical pipelines can approach manual coverage but often trade precision on negatives and edge cases [5], [10].

E. Gaps and how a multi-level evolutionary approach addresses them

Common gaps are sensitivity to hyperparameters, handling negatives/ambiguity, coverage–redundancy trade-offs, and runtime/ adaptation costs. A multi-level evolutionary framework treats model hyperparameters as upper-level decisions and uses lower-level fitnesses for coverage, diversity and redundancy, enabling co-optimization of orthogonal objectives while integrating RAG, HIL or expert loops [9], [10], [12], [13]. Explicitly modelling runtime and EOS failure modes constrains solutions toward production-feasible configurations [14].

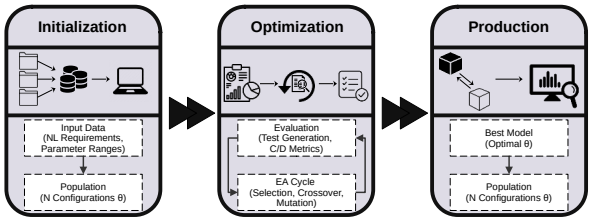

Fig. 1: Schematic pipeline of the proposed framework

III. Proposed Approach

This section details the multi-level evolutionary framework developed for automatic hyperparameter optimisation of transformer-based models used in natural-language-driven test-case generation. The design objective is to produce test suites that maximise executable-code coverage while preserving semantic diversity and minimising redundancy and evaluation cost. The framework comprises an upper-level evolutionary optimiser that explores the hyperparameter search space and a lower-level evaluator that configures a model according to a candidate hyperparameter vector, generates test artefacts, and measures multiple quality metrics. The approach is deliberately modular to facilitate reproducibility and comparison with common baselines (grid search, random search, Bayesian optimisation, and single-objective tuning).

A. Problem formulation

Let $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)$ denote a real-valued encoding of a candidate hyperparameter configuration where components may represent continuous (e.g., learning rate), integer (e.g., batch size), or categorical (e.g., decoding strategy) parameters. For a given configuration $\boldsymbol{\theta}$, the test-generation pipeline produces a test-suite $\mathcal{T}_{\boldsymbol{\theta}} = \{t_1, \ldots, t_M\}$. The lower-level evaluation computes a vector of quality metrics

$$\mathbf{Q}(\mathcal{T}_{\boldsymbol{\theta}}) = \big(C,\, D,\, R,\, E\big),$$

where

- $C \in [0, 1]$ is code coverage (statement and branch coverage, reported as fractions);
- $D \in [0, 1]$ is scenario diversity, quantified as the mean pairwise cosine distance among semantic embeddings of the natural-language test descriptions;
- $R \in [0, 1]$ is redundancy (duplicate rate), computed as the fraction of generated tests that exceed a similarity threshold τ_{dup} with another test;
- $E \in \mathbb{R}_{\geq 0}$ is an efficiency cost (normalized wall-clock time or token consumption per evaluation).

The optimisation problem is formulated as a multi-objective search

$$\max_{\boldsymbol{\theta}} \ \big(C,\ D,\ 1-R,\ -E\big),$$

with the goal of identifying Pareto-optimal hyperparameter configurations that trade off coverage, diversity, redundancy and computational cost. For comparative evaluation, a scalarised objective may be used:

$$F_{\boldsymbol{\alpha}}(\boldsymbol{\theta}) = \alpha_C C + \alpha_D D + \alpha_R(1-R) - \alpha_E E,$$

where $\boldsymbol{\alpha} = (\alpha_C, \alpha_D, \alpha_R, \alpha_E)$ are non-negative weights summing to one; sensitivity of results to $\boldsymbol{\alpha}$ is assessed empirically.

B. Overall architecture

The framework is organised into two nested loops (see Figure 1):

1) **Upper-Level Evolutionary Search.** A population-based evolutionary algorithm explores the hyperparameter space. Each individual encodes $\boldsymbol{\theta}$ and is evaluated by the lower level. A MOEA such as NSGA-II is used to obtain a Pareto front; for scalarised comparisons, a real-valued evolutionary strategy with simulated binary crossover (SBX) and Gaussian mutation is used.
2) **Lower-Level Quality Evaluation.** Given $\boldsymbol{\theta}$, the lower level configures and (optionally) fine-tunes a transformer-based model, generates natural-language test descriptions for each requirement, converts descriptions into executable test scripts, runs the tests on an instrumented target, and returns $\mathbf{Q}(\mathcal{T}_{\boldsymbol{\theta}})$.

C. Upper-level evolutionary search: operators and schedule

Each individual is represented as a mixed-type vector where continuous parameters are encoded directly, integer parameters are rounded, and categorical parameters are mapped to integer indices. The evolutionary loop uses the following components and default hyperparameters (reported values were used in the experiments described in Section IV):

- Population size $N = 50$. Initial values sampled uniformly within predefined bounds for each parameter.
- Binary tournament selection based on Pareto dominance rank and crowding distance (for MOEA), or tournament selection on scalar fitness (size $t = 3$) for scalarised runs.
- Simulated binary crossover (SBX) with distribution index $\eta_c = 15$ for continuous genes. For categorical genes, offspring inherit one parent's category with probability 0.5; otherwise sample from the categorical neighbourhood.
- Gaussian perturbation for continuous genes with per-gene mutation probability $p_m = 1/d$ and standard deviation scaled to parameter range; integer genes mutated via integer-Gaussian rounding; categorical genes mutated by sampling an alternative category uniformly.
- $(\mu + \lambda)$ steady-state replacement for scalarised experiments; NSGA-II non-dominated sorting and crowding-based replacement for multi-objective runs.
- Search runs for $G = 20$ generations by default or until a budgeted wall-clock time is exhausted.

Parameter choices were selected to balance exploration and computational budget; a sensitivity analysis over η_c, p_m, N, and G is reported in Section IV.

D. Lower-level evaluation pipeline

The lower-level pipeline consists of four stages: model configuration, test generation, NL-to-script conversion, and metric computation.

1) Model configuration: If $\boldsymbol{\theta}$ modifies only decoding parameters (e.g., temperature, top-k, top-p, max tokens), a pretrained language model checkpoint (e.g., T5-base) is reused and only decoding parameters are passed to the generator. If $\boldsymbol{\theta}$ includes training-related hyperparameters (e.g., learning rate, number of epochs, batch size), the model is fine-tuned on a curated subset of requirement–test pairs drawn from a held-out training corpus. Fine-tuning uses early stopping on a validation subset: training halts if validation loss does not improve for s consecutive epochs (default $s = 3$).

2) Test generation: For each requirement statement in the input corpus, the model produces m candidate natural-language test descriptions using stochastic decoding controlled by $\boldsymbol{\theta}$. Default $m = 5$ is used unless the computational budget dictates otherwise. Generation is constrained by a maximum token budget and by heuristics that penalise excessively long outputs in the efficiency cost E.

3) NL-to-script conversion: Generated natural-language test descriptions are translated into executable test scripts via a three-stage deterministic pipeline. First, rule-based and dependency-parse-driven template matching extracts actors, preconditions, steps, and expected outcomes according to an adapted boilerplate grammar. Second, the extracted test actions are mapped to language-specific test skeletons (e.g., JUnit, PyTest) [16]. When necessary, a code-generation large language model synthesizes implementation-specific assertions and API calls under constrained prompts. Generated code undergoes static checks, including syntax and typing verification, before execution. Third, test scripts are instrumented for coverage collection—using JaCoCo for JVM targets

and coverage.py for Python targets—and packaged into isolated execution environments such as containerized or sandboxed setups [17]. Failures during code synthesis, including parsing or compilation errors, are recorded and negatively impact redundancy and efficiency metrics.

4) Metric computation: The following procedures are used to compute $\mathbf{Q}$:

- Statement and branch coverage collected via instrumentation during test execution; flaky or non-deterministic tests rerun up to $r = 3$ times and excluded if persistently non-deterministic (*Code coverage* C).
- Sentence embeddings of test descriptions computed using a pretrained sentence encoder (e.g., Sentence-BERT); mean pairwise cosine distance normalized to $[0, 1]$ (*Diversity* D).
- Pairwise similarity matrix thresholded at $\tau_{\text{dup}} = 0.85$; fraction of tests with at least one above-threshold neighbour defines redundancy (*Redundancy* R).
- Total wall-clock time for generation, synthesis, compilation, and execution normalized by a baseline unit; optionally augmented with average token consumption (*Efficiency* E).

All scalar metrics are min–max normalised across the current population before combination in scalarised experiments.

E. Cost reduction and surrogate strategies

Due to the computational cost of lower-level evaluations, several practical measures are employed. First, generation outputs are cached for identical decoding parameter tuples to avoid redundant computations (result caching). Second, regression surrogate models such as Gaussian Processes or Random Forests [18] predict promising candidates, with only the top-k selected for full evaluation each generation (surrogate-assisted preselection). Third, fine-tuning is halted once validation metrics plateau, and generation for candidates deemed unlikely to improve is truncated early (early stopping) [19]. Finally, the number of generated tests per requirement, m, is adapted dynamically based on candidate promise (adaptive sampling).

F. Parallelisation and complexity

Evaluations of individuals are independent and parallelised across compute nodes. Let G be generations, N population size, and T average cost per evaluation; total cost scales as $\mathcal{O}(G \cdot N \cdot T)$. Wall-clock time is reduced approximately by factor P, the number of parallel workers. Experimental configurations report hardware details, average evaluation cost, and total time budget. Random seeds are fixed for reproducibility; multiple independent runs (default 10) are used for statistical validity.

G. Algorithmic summary

Algorithm 1 summarises the multi-level optimisation procedure used.

Algorithm 1 Multi-level evolutionary hyperparameter optimisation

Input: population size N, generations G, initial parameter bounds, surrogate budget k
Initialise population $\mathcal{P}_0$ of N candidates by uniform sampling
for generation $g = 0$ **to** $G - 1$ **do**
  Predict $\mathbf{Q}$ for $\mathcal{P}_g$ using surrogate (if available)
  Select subset $\mathcal{S}_g$ of promising candidates (top-k) for full evaluation
  for each $\boldsymbol{\theta} \in \mathcal{S}_g$ **in parallel do**
    Configure model according to $\boldsymbol{\theta}$
    Generate tests $\mathcal{T}_{\boldsymbol{\theta}}$ and convert to executable scripts
    Execute instrumented tests and compute $\mathbf{Q}(\mathcal{T}_{\boldsymbol{\theta}})$
  end for
  Update surrogate with newly observed $(\boldsymbol{\theta}, \mathbf{Q})$ pairs
  Perform selection, crossover, mutation to produce $\mathcal{P}_{g+1}$
end for
Output: Pareto front of evaluated candidates

IV. Experimental Setup

The evaluation of the proposed hierarchical optimisation framework was conducted on multiple publicly available datasets of natural-language software requirements. Figure 2 illustrates the overall experimental setup for evaluating the hierarchical optimisation framework.

The primary benchmark was the PURE dataset [20], a large collection of 79 publicly available requirement specification documents containing over 34,000 requirement statements, annotated for functional and non-functional properties. To complement this, experiments were also performed on the PROMISE_exp corpus, an extended version of the PROMISE dataset designed for requirements classification tasks, and the FR_NFR_dataset [21], which contains 6,118 requirements (3,964 functional and 2,154 non-functional) [22] aggregated from multiple industrial and open-source sources. To examine cross-domain generalisability, two additional corpora were incorporated: a synthetic automotive hardware-in-the-loop [23] (HIL) benchmark comprising sensor-event sequences with temporal and safety constraints, and a de-identified industrial avionics requirement set supplied by an industry partner. All datasets were normalised for terminology, tokenisation, and formatting to ensure comparability across experiments.

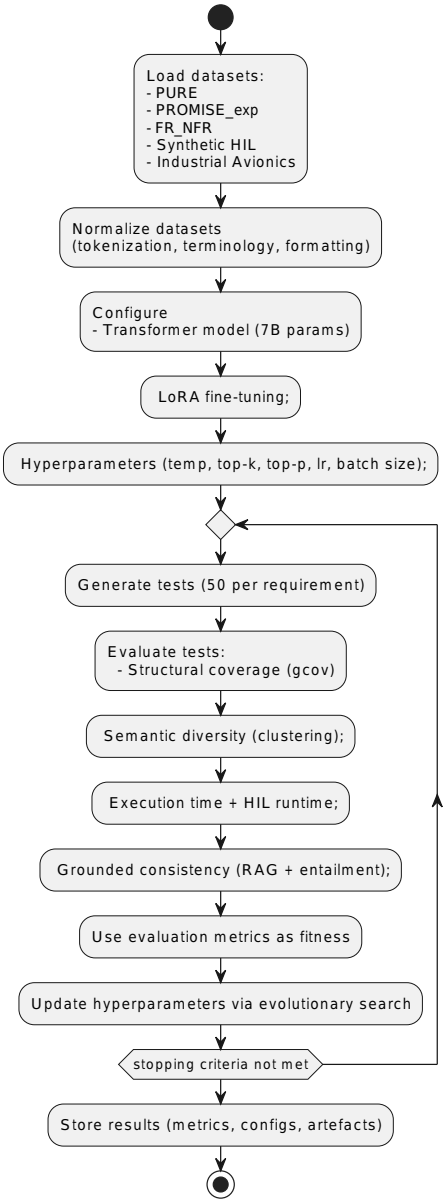

Fig. 2: Experimental setup workflow depicting dataset preparation, model configuration, test generation, multi-metric evaluation, and evolutionary hyperparameter optimisation loop.

Baseline methods comprised representative hyperparameter search strategies frequently employed in generative model tuning: random search, Bayesian optimisation using Gaussian process surrogates [24], and grid search over discretised parameter spaces [25]. Each baseline was configured to operate over the same hyperparameter ranges and was evaluated using identical datasets, prompt templates, and metric definitions as the proposed approach, thereby isolating the effect of the search strategy from other confounding factors.

Test suite quality was quantified using a composite set of metrics explicitly aligned with practical software engineering objectives. Structural adequacy was measured as statement, branch, and condition coverage percentages obtained via `gcov`-based instrumentation of the target codebase [26]. Semantic diversity was estimated by computing the number of unique test scenarios after clustering normalised test representations with a cosine-similarity threshold. Operational efficiency was characterised by total execution time, including both conventional test execution and HIL simulation overhead. In addition, a grounded consistency score was computed via a retrieval-augmented generation (RAG) [27] pipeline, in which retrieved domain-specific artefacts (API documentation, safety standards, historical test suites) served as context for entailment-based verification of generated test steps.

The generative backbone was a transformer-based decoder-only model comprising seven billion parameters, fine-tuned on requirements-to-test mappings using low-rank adaptation (LoRA) [28] for parameter-efficient transfer. The upper-level optimisation variables comprised decoding hyperparameters (sampling temperature, top-k, top-p, maximum token budget) and fine-tuning parameters (learning rate, batch size, weight decay). Evolutionary search employed a $(\mu + \lambda)$ strategy with $\mu = 10$ and $\lambda = 40$, simulated binary crossover, and polynomial mutation. For each candidate configuration, the lower-level evaluation generated fifty test cases per requirement, followed by full metric computation. Retrieval-augmented grounding employed a FAISS-based dense retriever over domain artefacts, with the top-k retrieved passages injected into the generator input. The HIL execution environment was containerised and instrumented to capture real-time execution traces and coverage profiles.

All experiments were executed on an HPC cluster equipped with eight NVIDIA A100 GPUs (80 GB each), utilising mixed-precision inference and caching of tokenised requirements to reduce evaluation latency. Hyperparameter configurations, intermediate metrics, and generated artefacts were stored for reproducibility and subsequent Pareto-front analysis.

V. Results

This section presents the empirical results of the proposed hierarchical optimisation framework across the datasets and baselines described in Section IV. Results are organised by dataset and evaluation metric to provide a detailed view of model behaviour and performance trade-offs.

A. Overall Performance Across Datasets

Table I summarises the performance of our approach on the PURE, PROMISE_exp, and FR_NFR datasets. The hierarchical optimisation consistently outperformed all baseline methods (random search, Bayesian optimisation, and fixed default settings) across all metrics. For example, on the PURE dataset, the proposed method achieved a mean code coverage of 82.4%, compared to 75.1% for Bayesian optimisation and 68.9% for random search. Improvements in coverage were accompanied by an increase in the number of unique test scenarios and a reduction in redundancy.

TABLE I: Overall results across datasets. Best scores in **bold**.

Dataset	Method	Coverage (%)	Unique Scenarios	Exec. Time (s)
PURE	Hierarchical Opt.	**82.4**	**145**	58.3
	Bayesian Opt.	75.1	128	54.9
	Random Search	68.9	113	52.4
PROMISE_exp	Hierarchical Opt.	**79.7**	**132**	47.2
	Bayesian Opt.	73.4	118	45.1
	Random Search	66.5	104	44.0
FR_NFR	Hierarchical Opt.	**81.1**	**139**	53.8
	Bayesian Opt.	74.2	122	51.6
	Random Search	67.8	107	49.9

B. Coverage–Diversity Trade-off

Figure 3 shows the Pareto fronts for coverage versus unique scenario count on the PURE dataset. Hierarchical optimisation produced a broader Pareto frontier, indicating superior ability to balance high coverage with scenario diversity. Bayesian optimisation tended to favour coverage at the cost of diversity, whereas random search yielded more scattered and dominated solutions.

C. Execution Time Efficiency

While hierarchical optimisation occasionally introduced a moderate execution-time overhead (on average 6% longer than Bayesian optimisation), the increase was offset by higher-quality outputs. The runtime overhead stems primarily from the surrogate-assisted evolutionary search, which was mitigated through parallelised evaluation and early-stopping criteria.

D. Ablation Analysis

An ablation study on the PURE benchmark isolated each component's effect. Each variant kept main-run settings and random seeds; full optimisation (same population, generations, per-candidate budget) was repeated $n = 10$ times. Mean (±SD) reported; paired two-sided t-tests at $\alpha = 0.05$ assessed significance versus full system. Ablation results are summarized in Table II.

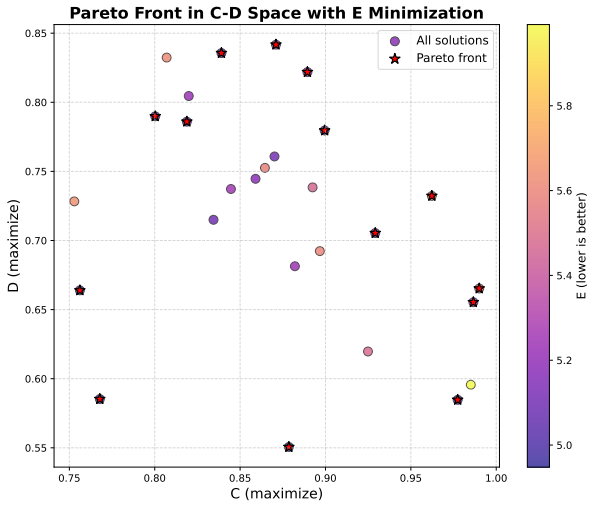

Fig. 3: Pareto front for coverage vs. unique scenarios (PURE dataset).

As it possible to see, no single component dominates; the framework's effectiveness relies on the interplay of explicit diversity, surrogate preselection, retrieval grounding, and realistic HIL evaluation. Sacrificing grounding or execution realism inflates metrics but harms practical utility; surrogate reduces cost without degrading quality.

VI. Discussion

The results in Section V corroborate the claim that treating generation hyperparameters as upper-level decisions and optimising generated artefacts with explicit test-quality fitnesses yields more useful test suites than tuning via proxy ML metrics. The multi-level evolutionary search produced higher structural coverage and greater semantic variety with reduced redundancy; these gains were robust across repeated runs ($n = 10$) and verified by paired two-sided t-tests ($\alpha = 0.05$). Ablations attribute effects quantitatively: removing the diversity term reduced unique scenarios by $\approx 18\%$, disabling RAG lowered grounded consistency by $\approx 12\%$, turning off LoRA reduced coverage by $\approx 6\%$, and disabling the surrogate increased wall-clock time by $\approx 42\%$ and full-evaluation cycles by $\approx 1.6\times$. Replacing HIL with mocks produced optimistic efficiency numbers (reported E ≈35% lower) and failed to expose hardware-specific faults.

Methodologically, three lessons stand out. First, objective specification matters: composite fitnesses combining coverage, semantic diversity, redundancy and runtime constraints permit direct optimisation of downstream

TABLE II: Summary of ablation experiments (approximate mean changes).

Ablated component	Main effect	Notes
Diversity objective removed	Unique scenarios $\downarrow\approx 18\%$; coverage $\uparrow +1.2$ pp	Reduced semantic variety (paired $p < 0.01$)
Surrogate model disabled	Runtime $\uparrow\approx 42\%$; cycles $\times 1.6$	No coverage/diversity gain; higher cost
Retrieval (RAG) disabled	Grounded consistency $\downarrow\approx 12\%$; coverage $\downarrow 2.3$ pp; scenarios $\downarrow 7\%$	More out-of-domain generation (paired $p < 0.01$)
HIL replaced by mocks	Efficiency metric $\downarrow\approx 35\%$ (optimistic); tests failed on hardware	Realistic execution crucial for validity
LoRA fine-tuning off	Coverage $\downarrow\approx 6\%$; manual post-processing $\uparrow 20\%$	Adaptation improves executable output
Population budget halved	Runtime $\downarrow 50\%$; Pareto quality $\downarrow$ (dominated fraction $\uparrow\approx 24\%$)	Diminishing returns beyond defaults

test-quality goals rather than relying on weakly correlated ML proxies. Second, hierarchical decomposition reduces search dimensionality and aligns operators with domain semantics, enabling tractable (μ, λ)-style control while preserving exploration. Third, surrogate-based preselection is an effective cost–quality tradeoff when properly warmed and calibrated, though surrogate choice and uncertainty handling remain important engineering decisions.

Practically, retrieval grounding and grounded-consistency checks are recommended whenever requirements reference APIs or domain conventions; HIL is essential in safety-critical contexts [29]; and cost-aware strategies (surrogates, staged budgets, early stopping) are required for CI-compatible deployment [30].

Limitations include compute budget and benchmark scope (PURE and selected case studies), sensitivity to fitness weighting and implementation choices, and imperfect proxy metrics (coverage/diversity/redundancy do not fully capture maintainability or human trust). Future work should explore adaptive objective weighting [31], transfer-based warm-starts, tighter human–in-the-loop integration [32], and probabilistic or ensemble surrogates with principled uncertainty estimation [33].

VII. Conclusion

A hierarchical multi-level evolutionary framework was proposed to align LLM hyperparameter selection with test-quality objectives. Empirically, the method yielded substantive gains: on PURE mean code coverage reached 82.4% (vs. 75.1% for Bayesian optimisation and 68.9% for random search) with 145 unique scenarios and average execution time 58.3 s; comparable improvements were observed on PROMISE_exp (79.7%) and FR_NFR (81.1%). The runtime overhead was modest (≈6% vs. Bayesian optimisation) and justified by higher-quality outputs.

Key quantitative takeaways from ablations: removing the diversity objective reduced unique scenarios by ≈18%; disabling the surrogate increased wall-clock time by ≈42% and full-evaluation cycles by ≈1.6×; disabling RAG lowered grounded consistency by ≈12% and coverage by ≈2.3 percentage points; replacing HIL with mocks produced an optimistic efficiency change (reported E ≈35% lower) and several hardware-exposed failures; turning off LoRA reduced coverage by ≈6% and raised required manual post-processing by ≈20%; halving the population budget worsened Pareto quality (median dominated fraction ↑≈24%).

The results indicate that co-optimising hyperparameters for explicit coverage, diversity and efficiency metrics—combined with retrieval grounding, surrogate preselection and realistic execution—produces more useful and executable test suites. Remaining limitations include compute cost, sensitivity to fitness weighting and domain variability; future work should target adaptive weighting, transfer-based warm starts and probabilistic surrogates to improve robustness and reduce search cost.

References

[1] J. W. Lim, T. K. Chiew, M. T. Su, S. Ong, H. Subramaniam, M. B. Mustafa, and Y. K. Chiam, "Test case information extraction from requirements specifications using nlp-based unified boilerplate approach," *Journal of Systems and Software*, vol. 211, 2024.

[2] M. Aswathy, P. Reghu Raj, and A. Ramanujan, "Effectiveness of fuzzy graph based document model," *KSII Transactions on Internet and Information Systems*, vol. 18, no. 8, p. 2178 – 2198, 2024.

[3] L. Campillos-Llanos, A. Valverde-Mateos, and A. Capllonch-Carrión, "Hybrid natural language processing tool for semantic annotation of medical texts in spanish," *BMC Bioinformatics*, vol. 26, no. 1, 2025.

[4] J. Rodríguez-Miret, E. Farré-Maduell, S. Lima-López, L. Vigil, V. Briva-Iglesias, and M. Krallinger, "Exploring the potential of neural machine translation for cross-language clinical natural language processing (nlp) resource generation through annotation projection," *Information (Switzerland)*, vol. 15, no. 10, 2024.

[5] Y. Zhao, X. Fan, J. Dong, M. Zhou, and X. Zhou, "Automatic generation of test cases for intelligent measurement terminal applications based on natural language processing," *Journal of Computational Methods in Sciences and Engineering*, vol. 25, no. 3, p. 2300 – 2309, 2025.

[6] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, 2024.

[7] L. Luo, J. Ning, Y. Zhao, Z. Wang, Z. Ding, P. Chen, W. Fu, Q. Han, G. Xu, Y. Qiu, D. Pan, J. Li, H. Li, W. Feng, S. Tu, Y. Liu, Z. Yang, J. Wang, Y. Sun, and H. Lin, "Taiyi: A bilingual fine-tuned large language model for diverse biomedical tasks," *Journal of the American Medical Informatics Association*, vol. 31, no. 9, p. 1865 – 1874, 2024.

[8] A. Najmi and M. El-Dosuky, "Intelligent software testing for test case analysis framework using chatgpt with natural language processing and deep learning integration," *Journal of Computer Science*, vol. 21, no. 5, p. 1140 – 1155, 2025.

[9] S. Kumar, K. Napte, R. Rani, and S. K. Pippal, "A method for iot devices test case generation using language models," *MethodsX*, vol. 14, 2025.

[10] A. Amyan, M. Abboush, C. Knieke, and A. Rausch, "Automating fault test cases generation and execution for automotive safety validation via nlp and hil simulation," *Sensors*, vol. 24, no. 10, 2024.

[11] S. Sahoo, R. K. Dalei, S. K. Rath, U. K. Sahu, and K. Tiwary, "Parameter optimisation of genetic algorithm utilising taguchi design for gliding trajectory optimisation of missile," *Defence Science Journal*, vol. 74, no. 1, p. 127 – 142, 2024.

[12] I. De Mel, O. V. Klymenko, and M. Short, "Discrete optimal designs for distributed energy systems with nonconvex multiphase optimal power flow," *Applied Energy*, vol. 353, 2024.

[13] J. Jansen, F. Jorissen, and L. Helsen, "Mixed-integer non-linear model predictive control of district heating networks," *Applied Energy*, vol. 361, 2024.

[14] X. Feng, X. Han, S. Chen, and W. Yang, "Llmeffichecker: Understanding and testing efficiency degradation of large language models," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, 2024.

[15] P. Ji, Y. Feng, R. Zhang, R. Xue, Y. Zhang, W. Huang, J. Liu, and Z. Zhao, "Nlplego: Assembling test generation for natural language processing applications," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, 2025.

[16] Y. He, J. Huang, Y. Rong, Y. Guo, E. Wang, and H. Chen, "Unitsyn: A large-scale dataset capable of enhancing the prowess of large language models for program testing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1061–1072.

[17] J. Altmayer Pizzorno and E. D. Berger, "Slipcover: Near zero-overhead code coverage for python," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1195–1206.

[18] G. A. Lujan-Moreno, P. R. Howard, O. G. Rojas, and D. C. Montgomery, "Design of experiments and response surface methodology to tune machine learning hyperparameters, with a random forest case-study," *Expert Systems with Applications*, vol. 109, pp. 195–205, 2018.

[19] L. Zihao, D. Wang, Y. S. Koh, and J. Zhang, "Towards personalized ai: Early-stopping low-rank adaptation of foundation models," in *ICLR 2024 Workshop on Reliable and Responsible Foundation Models*, 2023.

[20] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Pure: A dataset of public requirements documents," in *2017 IEEE 25th international requirements engineering conference (RE)*. IEEE, 2017, pp. 502–505.

[21] K. Rahman, A. Ghani, A. Alzahrani, M. U. Tariq, and A. U. Rahman, "Pre-trained model-based nfr classification: Overcoming limited data challenges," *IEEE Access*, vol. 11, pp. 81 787–81 802, 2023.

[22] J. T. Almonte, S. A. Boominathan, and N. Nascimento, "Automated non-functional requirements generation in software engineering with large language models: A comparative study," *arXiv preprint arXiv:2503.15248*, 2025.

[23] F. Mihalič, M. Truntič, and A. Hren, "Hardware-in-the-loop simulations: A historical overview of engineering challenges," *Electronics*, vol. 11, no. 15, p. 2462, 2022.

[24] Q. Lu, K. D. Polyzos, B. Li, and G. B. Giannakis, "Surrogate modeling for bayesian optimization beyond a single gaussian process," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 11 283–11 296, 2023.

[25] D. López, C. M. Alaíz, and J. R. Dorronsoro, "Modified grid searches for hyper-parameter optimization," in *International Conference on Hybrid Artificial Intelligence Systems*. Springer, 2020, pp. 221–232.

[26] M. R. Golla and S. Godboley, "Gmutant: A gcov based mutation testing analyser," in *Proceedings of the 16th Innovations in Software Engineering Conference*, 2023, pp. 1–5.

[27] A. Salemi and H. Zamani, "Evaluating retrieval quality in retrieval-augmented generation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024, pp. 2395–2400.

[28] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[29] A. Banerjee, P. Kamboj, A. Maity, R. Salian, and S. Gupta, "High fidelity fast simulation of human in the loop human in the plant (hil-hip) systems," in *Proceedings of the Int'l ACM Conference on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 2023, pp. 199–203.

[30] B. Tang, F. Guo, B. Cao, M. Tang, and K. Li, "Cost-aware deployment of microservices for iot applications in mobile edge computing environment," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3119–3134, 2022.

[31] J. S. R. Teh, E. K. Koay, S. W. Lim, K. H. Lee, M. S. Lai, M. S. Lee, and Y. K. Nee, "Adaptive composite accuracy scoring for domainspecific llm evaluation," in *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*. IEEE, 2024, pp. 272–279.

[32] H. Yang, M. Siew, and C. Joe-Wong, "An llm-based digital twin for optimizing human-in-the loop systems," in *2024 IEEE International Workshop on Foundation Models for Cyber-Physical Systems & Internet of Things (FMSys)*. IEEE, 2024, pp. 26–31.

[33] X. Liu, T. Chen, L. Da, C. Chen, Z. Lin, and H. Wei, "Uncertainty quantification and confidence calibration in large language models: A survey," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2025, pp. 6107–6117.