Evaluating Generative AI Outputs as a Human-Machine Interface Between Domain Experts and Domain-Specific Languages: A Cost-of-Error Framework

Panagiotis Kalogeropoulos[1][0009−0006−3490−8748]

High Tech Embedded Software Research Group, Fontys University of Applied Sciences, 5612 MA Eindhoven, The Netherlands
p@pkal.dev

Abstract. Large Language Models increasingly serve as Human-Machine Interfaces between domain experts and Domain-Specific Languages (DSL), enabling natural language translation to structured DSL code without extensive programming knowledge. Existing evaluation methods for that translation focus on syntactic correctness, assume zero deployment cost, and lack domain-specific context. In industrial settings, errors carry real operational consequences, and the evaluation methods of this translation need to reflect such criticalities. A generic Cost-of-Error framework is presented, that quantifies the operational and monetary impact of AI-generated DSL code through a multi-dimensional evaluation from configurable stakeholder perspectives. The framework is instantiated using an LLM-as-a-Judge architecture with six specialized domain expert judges. Validation occurs through a comparative study of three GPT-5 model variants generating MermaidJS sequence diagrams for the OpenRemote IoT platform. The results show that model capability correlates with evaluation scores (17% increase between smallest and biggest model), and the framework translates these assessments into actionable estimates, bridging abstract quality metrics and industrial deployment requirements.

Keywords: Generative AI · Domain-Specific Languages · Human-Machine Interface · Cost-of-Error · LLM-as-a-Judge · Code Evaluation

1 Introduction

Large Language Models enable domain experts in fields such as industrial engineering and process control to convert domain knowledge from natural language into Domain-Specific Language (DSL) syntax [6]. This effectively positions LLMs as Human-Machine Interfaces between experts and DSL-based systems. In industrial contexts, this allows machine technicians and process engineers to generate diagrams, configure machinery, and describe system behaviors without extensive programming knowledge [3].

Despite this transformative potential, a critical challenge remains: **how can domain experts, with no programming knowledge, reliably evaluate the quality and safety of AI-generated DSL code before deployment?** Current evaluation methods suffer from key limitations, like restriction of focus on syntactic correctness while neglecting semantic accuracy and operational safety [8,9], assumption of no deployment cost, which is unrealistic when incorrect code causes physical damage or production downtime [2], full dependence on subjective human judgment that does not scale [10], and lack of domain-specific context such as physical constraints and safety thresholds [3].

To the best of our knowledge, no existing framework provides domain experts with quantifiable, multi-dimensional, and context-aware evaluation of AI-generated DSL code that accounts for operational consequences. The proposed Cost-of-Error framework addresses this gap.

2 Methodology

2.1 Cost-of-Error Framework

The framework is built on the principle that AI evaluation should be measured by operational consequences of deploying generated code. Cost-of-Error (CoE) is defined as the quantified operational and monetary impact resulting from deploying AI-generated DSL code in a target environment. Conceptually, for generated code c evaluated in environment S with context $\mathcal{C}$:

$$CoE(c, S, \mathcal{C}) = \sum_{i=1}^{n} w_i \cdot I_i(c, S, \mathcal{C}) \tag{1}$$

where I_i represents the impact on dimension i (e.g., safety violation cost, compliance penalty), w_i is an optional weight reflecting domain priorities, and n is the number of dimensions that are being used for evaluation.

In Equation 1, environment S denotes the target deployment system (e.g., a specific IoT platform instance with its hardware and software stack), while context C captures the operational profile: organizational size, user base, regulatory regime, and risk tolerance. Each I_i is computed from a named stakeholder perspective, such as a domain expert role with explicitly defined evaluation criteria (e.g., a "Cybersecurity Expert" evaluates authentication flows and injection surfaces). Weights w_i are set by the user of the framework to reflect deployment priorities.

The framework inherently defines two categories of evaluation dimensions: DSL-Specific Criteria (syntactic correctness, DSL conventions) and Domain-Specific Criteria (evaluation from the perspective of stakeholders affected by the output). This architecture is domain-agnostic: practitioners select judges appropriate to their deployment context. DSL-Specific Criteria can be evaluated using deterministic tooling (parsers, AST validators) as a pass/fail gate. Domain-Specific Criteria are more heavily based on subjective criteria, and are thus more difficult to impose gates for.

To provide meaningful monetary cost estimates, the framework introduces Ownership Contexts, that characterize the operational environment (e.g., company size, downtime cost, regulatory exposure) of the deployed code. These contexts are provided to judges, enabling severity scores to be contextualized appropriately, and are then translated into monetary impact ranges. The monetary Cost-of-Error value is generated using the per-dimension impact scores explained above, along with the defined Ownership Context used. Currently, an LLM is given all relevant information and is asked to generate its own arbitrary cost of error value using that Ownership Context.

Using all the above calculated values and LLM output, the LLM produces comprehensive test reports synthesizing all evaluation dimensions into an aggregate performance score, identified strengths and weaknesses, risk classification, monetary Cost-of-Error estimates, and a final acceptance verdict (Acceptable / Acceptable with Caution / Not Suitable). These reports enable human domain experts without the relevant programming or engineering experience to make informed go/no-go decisions.

2.2 Experimental Setup

For illustration and verification purposes, to compute Cost-of-Error without human evaluation for each assessment, multiple domain-specialized "judge" LLMs [5] are deployed, each representing a distinct stakeholder perspective.

As an example, a cybersecurity judge LLM evaluating a device provisioning diagram identified that the LLM-under-Test omitted client certificate validation entirely, scoring it 0.32. The same scenario generated by a bigger LLM-under-Test, included certificate exchange steps but lacked rate-limiting controls, receiving 0.68. These critiques feed into the test report, where the ownership context (for example, a 15-employee agency with $5,000/hour downtime cost) translates the security gap into an estimated $10,000–$20,000 incident exposure.

The Cost-of-Error framework is not the means to illustrate the validity of the LLM-as-a-Judge technique. The proposed framework itself is the evaluation architecture, while the integration of LLM judges is just one pluggable evaluation mechanism, that provides us the means to prove the applicability of the proposed framework. The validation presented here tests the framework's ability to discriminate between models of known different capability, which it achieves. This validity does not require absolute quantitative accuracy of individual judge scores, but it intends to illustrate the correct model order in terms of capability, which is evident across all three models tested. Establishing absolute calibration against human experts remains future work.

3 Results

The Cost-of-Error framework was instantiated for MermaidJS sequence diagram generation, describing software architectures for the OpenRemote open-source IoT platform[1]. Six domain expert judges were defined by creating their

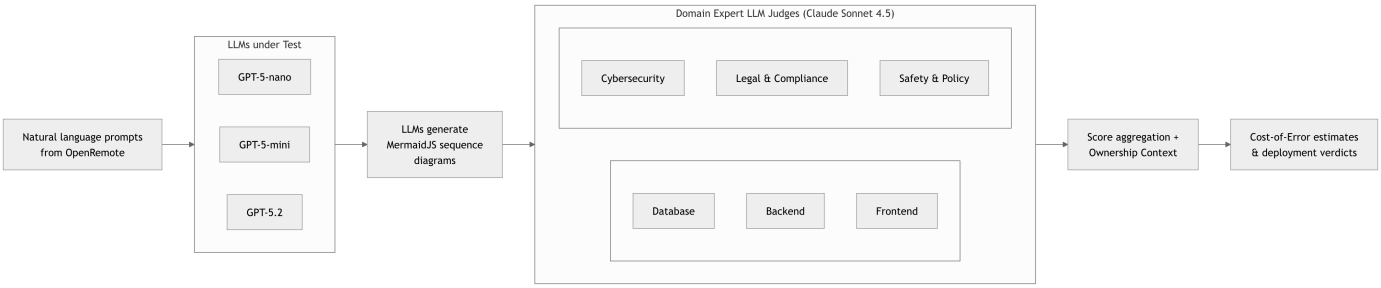

Fig. 1. Overview of the Cost-of-Error LLM comparison experimental setup

Table 1. Domain expert scores by model (0.0–1.0 scale) and Cost-of-Error estimates for a small software agency context.

Criterion / Metric	**GPT-5-nano**	**GPT-5-mini**	**GPT-5.2**
Database Expert	0.52	0.42	0.62
Backend Expert	0.32	0.62	0.82
Frontend Expert	0.52	0.68	0.58
Cybersecurity Expert	0.32	0.42	0.68
Legal & Compliance	0.72	0.72	0.68
Safety & Policy	0.68	0.72	0.72
Mean Score	**0.513**	**0.597**	**0.683**
Total CoE Range	$85k–255k	$45k–140k	$30k–55k
Verdict	Not Suitable	With Caution	With Caution

prompts. Three GPT-5 model variants (5.2, mini, and nano) served as LLMs under Test, with Claude Sonnet 4.5 as the judge model. The test dataset was constructed using a hybrid pipeline combining manually curated seed examples with real OpenRemote GitHub issues, synthetically generated with LLM-based expansion (Claude Sonnet 4.5), following a methodology similar to Shbita et al. [11]. Figure 1 illustrates the LLM evaluation pipeline.

Mean scores increase with model capability (17-percentage-point spread). Technical criteria (Backend: 0.32–0.82; Cybersecurity: 0.32–0.68) show the highest variance, while compliance and safety dimensions remain stable (0.68–0.72) across model sizes. Cost-of-Error for the smallest model is approximately 3× higher than the largest. Even GPT-5.2 achieved only 0.683 mean score, confirming that all current models require human review before production deployment. An organization using this framework would select $GPT-5.2$ over its *nano* counterpart because the three-fold reduction in estimated Cost-of-Error ($30k–55k vs. $85k–255k) justifies the higher inference cost, while still identifying that backend architecture and security are the specific dimensions requiring mandatory human review.

4 Discussion and Conclusion

The Cost-of-Error framework shifts evaluation focus from abstract correctness to operational consequences, bridging AI capabilities and industrial requirements. The LLM-as-a-Judge instantiation demonstrates the framework's scalability, but the framework itself is architecturally agnostic. It supports human judges, hybrid panels, or automated verification tools. Combining LLM-based evaluation with formal verification [2] is a promising direction for stronger correctness guarantees.

The novelty lies not in the scoring arithmetic, but in the pipeline for establishing the score itself: chaining multi-perspective domain evaluators with deployment-context monetary value assignment, to produce actionable risk assessments, easily consumable by non-technical domain experts. Prior work on code evaluation, either stops at quality scores or assumes zero cost of deployment of the Software-Under-Test [8].

A key threat to validity is that LLM judges may share biases with the LLMs under test; future work should validate correlation with human expert assessments. Trust calibration research and co-audit principles [4] inform how the framework's outputs should be presented to decision-makers.

The framework is generalizable beyond sequence diagrams. Relevant applications include legal contract specification via Symboleo [12], industrial PLC code generation with safety constraints [3], among others.

These domains share the core requirement the Cost-of-Error framework addresses: translating abstract quality scores into domain-specific, monetarily quantified risk assessments that support responsible AI adoption. Trust, control, and responsibility will increasingly be shared between humans and AI systems, with frameworks like Cost-of-Error providing the foundation for appropriate trust calibration.

The implementation is publicly available in GitHub, available at [7].

Acknowledgments. The author would like to extend his appreciation to Prof. Herman Jurjus of the HTES lectoraat of Fontys University of Applied Sciences for his help and support as the scientific advisor to the project, and to Profs. Qin Zhao and Jan Dobbelsteen, coaches and assessors of the research conducted, as part of the Master of Applied IT program.

The author would like to thank the ICT department of Fontys University of Applied Sciences for their support and funding of this research.

Disclosure of Interests. The author has no competing interests to declare. Generative AI tools (Claude, GPT-5 variants) were used both as objects of study and as evaluation infrastructure, as described in the methodology.

References

1. openremote/openremote (Jan 2026), `https://github.com/openremote/openremote`, original-date: 2016-02-03T11:14:02Z

2. Councilman, A., Fu, D., Gupta, A., Wang, C., Grove, D., Wang, Y.X., Adve, V.: Towards Formal Verification of LLM-Generated Code from Natural Language Prompts (Jul 2025). https://doi.org/10.48550/arXiv.2507.13290, http://arxiv.org/abs/2507.13290, arXiv:2507.13290 [cs]
3. Fakih, M., Dharmaji, R., Moghaddas, Y., Araya, G.Q., Ogundare, O., Faruque, M.A.A.: LLM4PLC: Harnessing Large Language Models for Verifiable Programming of PLCs in Industrial Control Systems. In: Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice. pp. 192–203 (Apr 2024). https://doi.org/10.1145/3639477.3639743, http://arxiv.org/abs/2401.05443, arXiv:2401.05443 [cs]
4. Gordon, A.D., Negreanu, C., Cambronero, J., Chakravarthy, R., Drosos, I., Fang, H., Mitra, B., Richardson, H., Sarkar, A., Simmons, S., Williams, J., Zorn, B.: Co-audit: tools to help humans double-check AI-generated content (Oct 2023). https://doi.org/10.48550/arXiv.2310.01297, http://arxiv.org/abs/2310.01297, arXiv:2310.01297 [cs]
5. Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., Wang, S., Zhang, K., Lin, Z., Zhang, B., Ni, L., Gao, W., Wang, Y., Guo, J.: A survey on LLM-as-a-judge. The Innovation p. 101253 (Jan 2026). https://doi.org/10.1016/j.xinn.2025.101253, https://www.sciencedirect.com/science/article/pii/S2666675825004564
6. Jackson, I., Jesus Saenz, M., Ivanov, D.: From natural language to simulations: applying AI to automate simulation modelling of logistics systems. International Journal of Production Research **62**(4), 1434–1457 (Feb 2024). https://doi.org/10.1080/00207543.2023.2276811, https://www.tandfonline.com/doi/full/10.1080/00207543.2023.2276811
7. Kalogeropoulos, P.: Cost-of-Error Evaluation Framework Implementation (Jan 2026), https://github.com/pankalog/master-s1-paper-code, original-date: 2026-01-18T15:30:57Z
8. Liguori, P., Improta, C., Natella, R., Cukic, B., Cotroneo, D.: Who evaluates the evaluators? On automatic metrics for assessing AI-based offensive code generators. Expert Systems with Applications **225**, 120073 (Sep 2023). https://doi.org/10.1016/j.eswa.2023.120073, https://www.sciencedirect.com/science/article/pii/S0957417423005754
9. Mitsopoulou, A., Koutrika, G.: Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities
10. Nauta, M., Trienes, J., Pathak, S., Nguyen, E., Peters, M., Schmitt, Y., Schlötterer, J., Keulen, M.v., Seifert, C.: From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI. ACM Computing Surveys **55**(13s), 1–42 (Dec 2023). https://doi.org/10.1145/3583558, http://arxiv.org/abs/2201.08164, arXiv:2201.08164 [cs]
11. Shbita, B., Ahmed, F., DeLuca, C.: MermaidSeqBench: An Evaluation Benchmark for LLM-to-Mermaid Sequence Diagram Generation (Nov 2025). https://doi.org/10.48550/arXiv.2511.14967, http://arxiv.org/abs/2511.14967, arXiv:2511.14967 [cs]
12. Zitouni, M.N., Anda, A.A., Rajpal, S., Amyot, D., Mylopoulos, J.: Towards the LLM-Based Generation of Formal Specifications from Natural-Language Contracts: Early Experiments with Symboleo (Nov 2024). https://doi.org/10.48550/arXiv.2411.15898, http://arxiv.org/abs/2411.15898, arXiv:2411.15898 [cs]