Proceedings of the ASME 2026
International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
IDETC-CIE 2026
August 23–26, 2026, Houston, TX, USA

DETC2026-XXXX

RELIABLE NATURAL-LANGUAGE TO SYSMLV2 TRANSLATION VIA VALIDATOR-DRIVEN ITERATIVE REFINEMENT

Chance LaVoie[1,*], Eladio Andujar Lugo[1], Levent Burak Kara[1]

[1]Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA

ABSTRACT

Keywords: SysMLv2, model-based systems engineering, large language models, validator-in-the-loop, natural language to model translation, syntactic validity

1. INTRODUCTION

Model-Based Systems Engineering (MBSE) replaces document-centric workflows with model-centric engineering, where formal artifacts carry requirements, structure, behavior, and verification intent across the lifecycle [1–4]. By making the model the primary technical source, MBSE reduces cross-document reconciliation overhead and improves traceability under change [2, 5].

SysMLv2 strengthens this transition by standardizing a formal language with complementary textual and graphical representations and machine-readable artifacts under OMG [6–8]. In practice, the same model can be authored as text, rendered visually, and exchanged across tools with less ambiguity. Because SysMLv2 is formal and textual, model construction is scriptable.

This scriptability, combined with the recent rise of LLMs in engineering practice, creates a concrete opportunity for automated SysMLv2 modeling workflows. LLM-based pipelines can generate candidate models directly from natural-language inputs.

Software engineering provides encouraging precedent: controlled and field studies report substantial productivity gains from LLM-assisted generation, including 55.8% faster task completion, 26.08% higher weekly completed tasks in enterprise randomized deployments, and measurable increases in pull-request throughput [9–11]. These results suggest that integrating generation into everyday tool use can materially accelerate engineering iteration.

The analogous opportunity in MBSE is to automate low-value model authoring steps while increasing modeling velocity. If natural-language prompts can be converted into *production-validator–accepted* SysMLv2, teams can spend less effort on syntax correction and more effort on architecture and verification reasoning.

Motivated by this potential, recent SysMLv2 generation systems have moved beyond unreliable one-shot prompting toward structured generation workflows [12, 13]. SysTemp uses a template-first multi-agent design centered on a *TemplateGeneratorAgent* and a *ParserAgent*. A Jinja2-based template tool first builds a structured SysMLv2 skeleton, then a writer agent fills in content while parser feedback guides iterative correction [12]. By contrast, the 2025 *Computers in Industry* framework organizes generation as an agentic loop with a RAG-based context engine and a validation engine, with ANTLR-based grammar validation as the syntax gate [13]. In that framework, the context engine retrieves semantically similar natural-language prompt and SysML file pairs and exposes them to the LLM as context [13]. Both approaches rely on context-free grammar (CFG) parsing as the primary basis for syntactic validation and as the primary acceptance metric.

In parallel, SysMBench has established a common benchmark for evaluating natural-language–to–SysML generation quality, especially on semantic alignment criteria [14]. Read together, the current state of the art suggests that generating SysMLv2 from natural language that passes grammar-parser checks is feasible.

Grammar parsability guarantees structural conformance to SysMLv2 context-free syntax: valid token ordering, balanced delimiters, and production-rule-compliant declarations. It does *not* guarantee production validation acceptance, because industrial validators enforce additional model-wide constraints such as name resolution, type consistency, ownership/composition rules, multiplicity constraints, and cross-reference integrity. Therefore, parser-passing outputs provide no assurance that they will be accepted by a production modeling environment for downstream use, including visualization, semantic validation, simulation, and optimization. Without acceptance under a full industrial validator, such models remain unusable in practice. In an auxiliary repository demonstration, we show ten distinct cases that pass SysML ANTLR parsing but fail production validation, empiri-

*Corresponding author: chancel@cmu.edu
Documentation for `asmeconf.cls`: Version 1.45, February 21, 2026.

cally reinforcing this distinction [15]. The critical gap is reliable generation of *production-validator–accepted* SysMLv2 from arbitrary natural-language prompts.

To close this gap, we instantiate a generate–verify–refine workflow in which candidate SysMLv2 models are iteratively evaluated against a production validation backend and revised until acceptance. This paradigm is not novel in itself. It draws from established traditions in formal methods and inductive synthesis, where candidate artifacts are proposed, evaluated by an external oracle, and corrected using deterministic feedback from that oracle [16–19]. Our contribution is to operationalize this verification-driven pattern for natural-language–to–SysMLv2 generation under industry-facing acceptance criteria.

Related work in code generation has already demonstrated the technical value of iterative verifier-in-the-loop refinement. Wang et al. show that deterministic compiler diagnostics can be used as structured correction signals in multi-stage neural code generation, improving acceptance of generated programs under executable checks [20]. Grubišić et al. similarly use compiler feedback to iteratively refine LLM outputs, treating validator responses as an explicit control signal for subsequent generations [21]. Taken together, these works establish the core mechanism we rely on: generate a candidate, evaluate it with a deterministic backend, feed diagnostics back into the next iteration, and repeat until the validator reports no remaining errors.

We apply this principle directly to MBSE. Our framework places a production SysMLv2 syntax validator, SysIDE [22], in the loop as a deterministic oracle, and generation proceeds until zero-error acceptance is achieved under that validator. At convergence, the resulting model is production-ready and operational within the modeling environment—loadable, renderable, analyzable, and suitable for downstream verification workflows. By elevating production-validator acceptance to the convergence criterion, SysMLv2 generation moves from syntactic plausibility to deployable modeling infrastructure, establishing a practical foundation for Copilot-like support in MBSE workflows.

2. RELATED WORK — NEEDS A LOT OF WORK – CHANCES LOWEST PRIORITY

Our review focuses on prior efforts in automated SysML v2 generation, grammar-constrained structured synthesis, and validator-guided iterative refinement, with an emphasis on approaches addressing reliable syntactic correctness in low-data modeling languages.

2.1. LLM-Based SysML v2 Model Generation

Recent work has explored using large language models to generate SysML v2 models from natural language. Bouamra et al. [12] propose SysTemp, a multi-agent framework that decomposes generation into requirement extraction, template-based skeleton construction, and grammar-level parsing feedback. Two specialist agents structure the flow: a *TemplateGeneratorAgent* uses a Jinja2-backed template tool to construct a syntactically organized SysMLv2 skeleton, and a *ParserAgent* validates the generated text against formal SysMLv2 grammar and returns diagnostics to a writer agent for iterative repair [12]. This work demonstrates that structural scaffolding and agent decomposition improve grammar conformity in sparse-data settings, with syntactic acceptance anchored to context-free grammar (CFG) parsing. However, syntactic correctness is defined at the grammar level and convergence is reported empirically rather than enforced as a termination invariant.

Cibrián et al. [13] introduce a distinct agentic architecture in *Computers in Industry*: generation is orchestrated in an agentic loop with a RAG-based context engine and a validation engine (for grammar-level checking), using ANTLR-generated CFG parsing as the syntactic acceptance mechanism. Their context engine retrieves semantically similar natural-language and SysML exemplar pairs to condition generation [13]. Their approach reports 100% syntactic validity across 20 curated prompts under grammar-level parsing. While this represents a significant advance in structured generation reliability, correctness is enforced through grammar parsing rather than full production validation. Because grammar-level validation ensures context-free structural conformity but does not enforce full static semantics under a production modeling environment, grammar-valid models may still fail production validation in practice. Together, these efforts establish that iterative validation substantially improves syntactic success in SysML v2 generation. However, prior systems define correctness primarily at the grammar level and evaluate on limited scenario sets. For industry use, the unresolved gap is generalizable *production-validator–accepted* syntactic correctness across diverse prompts. An additional gap is retrieval dependence: pipelines that require semantically similar NL-SysML exemplars may be less generalizable when comparable examples are sparse or unavailable. Our work builds on these insights while shifting the correctness oracle from grammar parsing to a production SysMLv2 validator and scaling evaluation to benchmark-level scenarios.

2.2. Grammar-Constrained and Template-Based Structured Synthesis

Structured synthesis approaches aim to reduce hallucinations in LLM output by constraining generation via templates or grammar rules. SysTemp [12] explicitly uses a template generator based on Jinja2 to construct syntactically compliant model skeletons prior to completion. Grammar-constrained decoding and post-generation parsing similarly reduce token-level structural invalidity.

While grammar validation ensures adherence to context-free rules, it does not enforce type resolution, cross-reference integrity, constraint satisfaction, or production modeling environment compatibility. Thus, grammar-valid artifacts may remain unusable within industrial MBSE workflows. Concretely, a model such as `port p: UndefinedType;` can be grammar-parsable yet fail production validation because the referenced type is unresolved; conversely, a missing delimiter (e.g., omitted semicolon) fails parsing before production validation semantics are even checked. Our approach differs by treating grammar conformity as necessary but insufficient and requiring full production validation acceptance prior to termination.

2.3. Validator-in-the-Loop and Verifier-Guided Generation

Prior work in neural code generation has explored leveraging compiler diagnostics to improve the compilability of model outputs. For example, Wang et al. [20] propose a multi-stage refinement framework that uses compiler feedback to iteratively revise generated programs and increase compilation success rates. Such approaches demonstrate that deterministic compiler signals can serve as effective supervisory feedback in programming-language settings. However, these systems operate in mature programming ecosystems and treat compilation success as an empirical objective rather than as a structural termination invariant.

Beyond compiler-feedback approaches in code generation, iterative generation guided by deterministic verifiers has also been studied under counterexample-guided inductive synthesis and execution-based refinement paradigms. Grubišić et al. [21] demonstrate that LLVM compiler feedback can serve as a supervisory signal for refining LLM-generated intermediate representation. Their work shows that deterministic compilation signals constrain output space and improve structural validity in programming languages.

While these approaches establish the feasibility of compiler-aware refinement in software domains, they differ in both objective and context from model-based systems engineering. Compiler feedback in such systems is typically used to improve optimization quality or increase compilation probability, rather than to enforce strict termination conditions. Moreover, these methods operate in programming languages with extensive training corpora and stable ecosystems.

In contrast, our work applies validator-in-the-loop refinement to SysMLv2, a newly standardized modeling language with sparse representation in LLM training data and strict production-validation requirements. Rather than treating validation results as heuristic feedback, we enforce zero-error acceptance under a production SysMLv2 validator (invoked via `syside check`) as a termination invariant. This elevates syntactic correctness from an empirical metric to a property guaranteed by construction, aligning generation reliability with production MBSE modeling-environment criteria rather than grammar-level approximations.

3. METHODOLOGY

3.1. Study Objective and Paired Design

This study evaluates one question: for the same prompt and the same model, does validator-in-the-loop refinement increase production validation acceptance relative to single-shot generation? The scope is strictly syntactic.

The experimental unit is one prompt–model pair. For each unit, we evaluate two paired conditions taken from the same run trajectory:

1. **Baseline (single-shot):** production validation outcome at iteration 1 only.

2. **Pipeline (iterative):** production validation outcome at the final available iteration after iterative repair.

Because both outcomes are taken from the same prompt–model run, this design isolates the effect of iterative validator feedback while holding prompt content and model identity fixed.

3.2. Validator-in-the-Loop Generation Procedure

Our controller follows a generate–validate–repair workflow for natural-language–to–SysMLv2 generation. At each iteration, the model proposes a complete SysMLv2 candidate, the production validator returns deterministic diagnostics, and the next model call is conditioned on those diagnostics.

Let P denote the natural-language prompt, M_t the generated candidate at iteration t, and $V(\cdot)$ the production validator. The update is

$$M_{t+1} = f(P, M_t, V(M_t)),$$

where $f(\cdot)$ is the model revision operator conditioned on validator feedback.

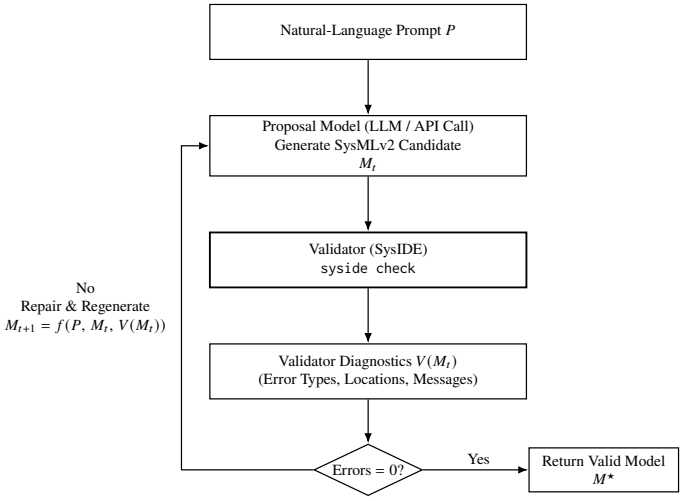

FIGURE 1: Validator-in-the-loop generate–validate–repair workflow. The prompt is fixed per case, and each revision is driven by deterministic validator diagnostics.

The production validator oracle is SysIDE validation (`syside check`) [22]. A run is successful and terminates only when zero validation errors are reported. This oracle choice is additionally supported by an auxiliary ten-case demonstration in the repository showing ANTLR parser pass with production-validation failure, i.e., grammar conformity without operational acceptability [15].

3.3. Dataset, Model Coverage, and Outcome Extraction

SysMBench provides paired natural-language prompts and ground-truth SysMLv2 models for benchmark evaluation [14]. In this study, we use the curated natural-language prompt set (IDs 1–151) as generation inputs because it was designed to stress SysMLv2 LLM generation across diverse modeling patterns.

To assess model-agnostic behavior of the same controller, we run four model configurations: OpenAI Codex 5.2 (`gpt-5.2-codex`) [23], Anthropic Sonnet 4.6 (`claude-sonnet-4-6`) [24], DeepSeek Reasoner (`deepseek-reasoner`) [25], and Mistral Large (`mistral-large-latest`) [26]. This yields 604 prompt-level cases (151 prompts × 4 models).

From saved run records, we extract first-shot and eventual pass/fail outcomes, iterations run, iterations to success, unresolved-within-cap status, first/final error counts, cumulative

error counts, per-iteration runtime, and token usage. Error families are grouped from validator diagnostics (for example, parsing and reference errors), while warnings are tracked separately and do not change pass/fail labels.

3.4. Endpoints and Statistical Analysis

The primary endpoint is production validation acceptance. We report first-shot pass rate, eventual pass rate, unresolved rate, absolute gain (percentage-point difference between eventual and first-shot pass rates), and relative gain:

$$\frac{p_{\text{final}} - p_{\text{first}}}{p_{\text{first}}}.$$

Because baseline and pipeline outcomes are paired binary observations on the same prompt–model unit, we test differences with McNemar's test using discordant counts (b, c) [27]. We use the exact two-sided variant when $b + c \leq 25$ and the continuity-corrected asymptotic variant otherwise, following matched-pairs guidance [28].

We summarize iterations-to-success with mean, median, standard deviation, quantiles, and maximum. Proportion confidence intervals are Wilson 95% score intervals [29]. Uncertainty in mean iterations-to-success is estimated with a bootstrap 95% confidence interval (10,000 resamples; fixed seed 20260220). These choices align with significance-focused reporting in recent LLM evaluation studies [30–33].

All claims are limited to syntactic production-validation acceptance. We do not infer semantic adequacy, behavioral correctness, or design quality from these outcomes.

3.5. Reproducibility

All code, run artifacts, and analysis outputs used in this study are stored in the project repository [34]. The repository includes the scripts required to regenerate campaign statistics, tables, and figures.

4. RESULTS

4.1. Overall Production-Validation-Gated Syntactic Outcomes

Across all 604 prompt-level trials (151 prompts for each of four models), first-shot production validation succeeded for 309/604 cases (51.16%; 95% Wilson CI: 47.18–55.13%). Under validator-in-the-loop refinement, eventual production validation succeeded for 604/604 cases (100.00%; 95% Wilson CI: 99.37–100.00%). The absolute gain from baseline single-shot to final pipeline output was 48.84 percentage points (relative gain 95.47%), with zero unresolved prompts.

A paired baseline-vs-pipeline test showed a strong shift toward success (McNemar: $b = 0$, $c = 295$, $p = 1.10 \times 10^{-65}$), indicating that improvements were driven by first-shot failures that were recovered by iterative repair.

4.2. Per-Model Syntactic Reliability

All models reached 151/151 eventual production validation under the validator-gated loop, but first-shot pass rates differed substantially. Anthropic Sonnet 4.6 had the highest first-shot pass rate (82.78%), while OpenAI (41.72%), DeepSeek Reasoner (41.06%), and Mistral Large (39.07%) showed similar first-shot behavior and correspondingly larger iterative gains.

4.3. Iteration and Recovery Dynamics

Iterations-to-success over all successful runs had mean 1.733, median 1, and maximum 11 (bootstrap 95% CI for mean: 1.654–1.816). The distribution was concentrated in early iterations: 309 cases succeeded at iteration 1, 201 at iteration 2, and 60 at iteration 3; only two cases required more than five iterations.

Among first-shot failures, recovery was complete: 295/295 (100%) failed-first-shot cases eventually passed production validation. For this recovery subset, mean iterations-to-success was 2.502 (median 2, max 11).

4.4. Production Validator Error Taxonomy

At first iteration, the dominant error families were `parsing-error` (1142), `reference-error` (559), and `port-definition-owned-usages-not-composite` (82). Across all iterations, the same families remained dominant, with cumulative counts 1658, 794, and 97 respectively.

4.5. Prompt-Level Difficulty and Diagnostic Burden

Error burden was heterogeneous across prompts. The largest pooled cumulative error volumes were observed for prompt IDs 93 (133), 9 (88), 139 (81), and 144 (80), indicating a long-tail of syntactically difficult cases even under eventual convergence.

4.6. Runtime and Token Sensitivity (Secondary)

Runtime and token data, where available in artifacts, indicate substantial efficiency differences by model despite identical syntactic endpoints. Mean wall-time ranged from 12.97 s (Mistral Large) to 90.45 s (DeepSeek Reasoner), and mean total tokens ranged from 1795.24 to 5938.80 per prompt.

These are secondary operational diagnostics and are not used as primary efficacy claims in this paper.

5. DISCUSSION

We discuss why deterministic validator diagnostics provide an effective supervision signal for new, sparsely represented languages like SysMLv2, and the resulting trade-offs in runtime and number of iterations.

6. CONCLUSION

We present a reproducible pathway for reliable natural-language to SysMLv2 translation by embedding deterministic production-validator feedback into LLM generation, thereby guaranteeing syntactically valid, production-validator–accepted output by construction.

APPENDIX A. AUXILIARY DEMONSTRATION: GRAMMAR PARSABILITY VS. PRODUCTION VALIDATION

To support the design choice of using production validation (rather than grammar parsing alone) as the acceptance oracle, we ran an auxiliary demonstration included in the repository under `experiments/antlr_vs_syside/` [15]. This demonstration is not a second primary experiment; it is a construct-validity check

showing that parser acceptance and production acceptance are distinct outcomes.

We evaluated 10 intentionally distinct SysMLv2 examples in `examples/mismatch_10_distinct/`. Each file was designed to remain grammar-parseable while violating a model-wide constraint typically enforced by a production validator. The evaluation pipeline was:

1. ANTLR parse check (third-party SysML parser integration in the repository).

2. Production validation check using SysIDE.

Results were unambiguous: all 10/10 examples passed ANTLR parsing, while 0/10 were accepted by production validation (10/10 mismatch cases). Validator diagnostics were dominated by unresolved-reference failures (9/10, `reference-error`), with one invocation-typing failure (1/10, `invocation-expression-instantiated-type`). This pattern directly illustrates that context-free syntax conformance is necessary but not sufficient for operational model acceptance in production modeling environments.

ACKNOWLEDGMENTS

REFERENCES

[1] Estefan, Jeff. "Survey of Model-Based Systems Engineering (MBSE) Methodologies." Technical Report No. INCOSE-TD-2007-003-02. INCOSE MBSE Initiative. 2007. URL https://www.incose.org/docs/default-source/ProductsPublications/SE-Resources/mbse/mbse-methodology-survey-rev-b.pdf.

[2] Madni, Azad M. and Sievers, Michael. "Model-Based Systems Engineering: Motivation, Current Status, and Research Opportunities." *Systems Engineering* Vol. 21 No. 3 (2018): pp. 172–190. DOI 10.1002/sys.21438. URL https://doi.org/10.1002/sys.21438.

[3] INCOSE. "Model-Based Systems Engineering (MBSE) Initiative." https://www.incose.org/products-and-publications/se-resources/mbse (2025). Accessed February 2026.

[4] SEBoK Editorial Board. "Model-Based Systems Engineering (MBSE)." https://sebokwiki.org/wiki/Model-Based_Systems_Engineering_(MBSE) (2025). Systems Engineering Body of Knowledge, accessed February 2026.

[5] INCOSE. "Systems Engineering Vision 2035." Technical report no. International Council on Systems Engineering. 2022. URL https://www.incose.org/docs/default-source/aboutse/se-vision-2035.pdf.

[6] Group, Object Management. "OMG Systems Modeling Language (SysML) v2 Specification, Version 1.0 Beta." Technical report no. Object Management Group (OMG). 2024. URL https://www.omg.org/spec/SysML/2.0/. Official release draft, September 2024.

[7] Object Management Group. "About SysML 2.0 (Normative and Machine-Readable Artifacts)." https://www.omg.org/spec/SysML/2.0/About-SysML (2025). Accessed February 2026.

[8] Object Management Group. "OMG Announces Formal Adoption of SysML v2." https://www.omg.org/news/releases/pr2025/07-21-25.htm (2025). Highlights textual and graphical notation and standard API/services layer; accessed February 2026.

[9] Peng, Sida, Kalliamvakou, Eirini, Cihon, Peter and Demirer, Mert. "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot." *arXiv preprint arXiv:2302.06590* (2023)DOI 10.48550/arXiv.2302.06590. URL https://arxiv.org/abs/2302.06590.

[10] Cui, Kevin Zheyuan, Peng, Sida, Quintana, Alexi and et al. "The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers." *Working Paper* (2025)URL https://economics.mit.edu/sites/default/files/inline-files/draft_copilot_experiments.pdf.

[11] Demirer, Mert, Peng, Sida, Quintana, Alexi and et al. "The Productivity Effects of Generative AI: Evidence from a Field Experiment with GitHub Copilot." *Working Paper* (2024)URL https://mit-genai.pubpub.org/pub/v5iixksv.

[12] Bouamra, A. et al. "SysTemp." arXiv:2506.21608 (2025). URL https://arxiv.org/abs/2506.21608.

[13] Cibrián, F. et al. "Agent-Based SysML v2 Synthesis with Retrieval-Augmented Generation and Grammar Validation." *Computers in Industry* (2025)DOI 10.1016/j.compind.2025.104350. URL https://doi.org/10.1016/j.compind.2025.104350.

[14] Jin, Dongming, Jin, Zhi, Li, Linyu, Fang, Zheng, Li, Jia, Chen, Xiaohong and Luo, Yixing. "A System Model Generation Benchmark from Natural Language Requirements." *arXiv preprint arXiv:2508.03215* (2025).

[15] LaVoie, Chance, Andujar Lugo, Eladio and Kara, Levent Burak. "ANTLR-versus-Production-Validation Demonstration for SysMLv2." https://github.com/cmuchancel/SysMBench_Compiler_In_Loop/tree/main/experiments/antlr_vs_syside (2026). Ten-case demonstration showing grammar-parser acceptance without production-validation acceptance. Accessed February 21, 2026.

[16] Clarke, Edmund M., Grumberg, Orna, Jha, Somesh, Lu, Yuan and Veith, Helmut. "Counterexample-Guided Abstraction Refinement." *Computer Aided Verification (CAV)*: pp. 154–169. 2000. DOI 10.1007/10722167_15.

[17] Solar-Lezama, Armando. "Program Synthesis by Sketching." Ph.D. Thesis, University of California, Berkeley. 2008.

[18] Jha, Susmit, Gulwani, Sumit, Seshia, Sanjit A. and Tiwari, Ashish. "Oracle-Guided Component-Based Program Synthesis." *International Conference on Software Engineering (ICSE)*: pp. 215–224. 2010. DOI 10.1145/1806799.1806833.

[19] Alur, Rajeev, Bodik, Rastislav, Juniwal, Garvit, Martin, Milo M. K., Raghothaman, Mukund, Seshia, Sanjit A., Singh, Rishabh, Solar-Lezama, Armando, Torlak, Emina and Udupa, Abhishek. "Syntax-Guided Synthesis." *Formal Methods in Computer-Aided Design (FMCAD)*: pp. 1–8. 2013. DOI 10.1109/FMCAD.2013.6679385.

TABLE 1: Ten-case auxiliary demonstration: all examples parse under ANTLR, none pass production validation.

Example ID	Injected Condition	ANTLR Parse	Production Validation
01	Missing imported namespace	Pass	Fail (`reference-error`)
02	Missing port type	Pass	Fail (`reference-error`)
03	Missing attribute type	Pass	Fail (`reference-error`)
04	Missing specialization base	Pass	Fail (`reference-error`)
05	Action typed by undefined behavior type	Pass	Fail (`reference-error`)
06	Invocation target is not a behavior	Pass	Fail (`invocation-expression-instantiated-type`)
07	Missing referenced feature in expression	Pass	Fail (`reference-error`)
08	Missing root-qualified namespace	Pass	Fail (`reference-error`)
09	Redefinition of missing feature	Pass	Fail (`reference-error`)
10	Missing namespace in type use	Pass	Fail (`reference-error`)

[20] Wang, Xin, Chen, Wenhu, Chen, Xinyun and Wang, William Yang. "Compilable Neural Code Generation with Compiler Feedback." *Findings of the Association for Computational Linguistics: ACL 2022*: pp. 138–150. 2022. URL https://aclanthology.org/2022.findings-acl.2/.

[21] Grubišić, A. et al. "Compiler Feedback for Large Language Models." arXiv:2403.14714 (2024). URL https://arxiv.org/abs/2403.14714.

[22] Sensmetry. "SysIDE: The Open-Source IDE and Compiler for SysML v2." https://sensmetry.com/syside/ (2024). Accessed November 2025.

[23] OpenAI. "GPT-5.2-Codex Model Documentation." https://developers.openai.com/api/docs/models/gpt-5.2-codex (2026). Model used in this campaign: `gpt-5.2-codex`. Accessed February 20, 2026.

[24] Anthropic. "Introducing Claude Sonnet 4.6." https://www.anthropic.com/news/claude-sonnet-4-6 (2026). API usage specifies `claude-sonnet-4-6`. Accessed February 20, 2026.

[25] DeepSeek. "Reasoning Model (`deepseek-reasoner`)." https://api-docs.deepseek.com/guides/reasoning_model (2026). Accessed February 20, 2026.

[26] Mistral AI. "Function Calling (Model Example: `mistral-large-latest`)." https://docs.mistral.ai/capabilities/function_calling/ (2026). Accessed February 20, 2026.

[27] McNemar, Quinn. "Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages." *Psychometrika* Vol. 12 No. 2 (1947): pp. 153–157. DOI 10.1007/BF02295996. URL https://doi.org/10.1007/BF02295996.

[28] Fagerland, Morten W., Lydersen, Stian and Laake, Petter. "The McNemar Test for Binary Matched-Pairs Data: Mid-p and Asymptotic Are Better Than Exact Conditional." *BMC Medical Research Methodology* Vol. 13 (2013): p. 91. DOI 10.1186/1471-2288-13-91. URL https://doi.org/10.1186/1471-2288-13-91.

[29] Wilson, Edwin B. "Probable Inference, the Law of Succession, and Statistical Inference." *Journal of the American Statistical Association* Vol. 22 No. 158 (1927): pp. 209–212. DOI 10.1080/01621459.1927.10502953. URL https://doi.org/10.1080/01621459.1927.10502953.

[30] Cheng, Zaiyu and Mastropaolo, Antonio. "An Empirical Study on the Effects of System Prompts in Instruction-Tuned Models for Code Generation." *arXiv preprint arXiv:2602.15228* (2026)DOI 10.48550/arXiv.2602.15228. URL https://arxiv.org/abs/2602.15228.

[31] Lee, Kye Hwa, Jang, Sujung, Kim, Grace Juyun, Park, Sukyoung, Kim, Doeun, Kwon, Oh Jin, Lee, Jae-Ho and Kim, Young-Hak. "Large Language Models for Automating Clinical Trial Criteria Conversion to Observational Medical Outcomes Partnership Common Data Model Queries: Validation and Evaluation Study." *JMIR Medical Informatics* Vol. 13 No. 1 (2025): p. e71252. DOI 10.2196/71252. URL https://medinform.jmir.org/2025/1/e71252.

[32] Wind, Sebastian, Sopa, Jeta, Truhn, Daniel, Lotfinia, Mahshad, Nguyen, Tri-Thien, Bressem, Keno, Adams, Lisa, Rusu, Mirabela, Köstler, Harald, Wellein, Gerhard, Maier,

Andreas and Arasteh, Soroosh Tayebi. “Multi-step Retrieval and Reasoning Improves Radiology Question Answering With Large Language Models.” *npj Digital Medicine* Vol. 8 No. 1 (2025): p. 790. DOI 10.1038/s41746-025-02250-5. URL https://doi.org/10.1038/s41746-025-02250-5.

[33] Roberts, Jonathan, Han, Kai and Albanie, Samuel. “GRAB: A Challenging GRaph Analysis Benchmark for Large Multimodal Models.” *arXiv preprint arXiv:2408.11817* (2024)DOI 10.48550/arXiv.2408.11817. URL https://arxiv.org/abs/2408.11817.

[34] LaVoie, Chance, Andujar Lugo, Eladio and Kara, Levent Burak. “SysMBench Compiler-in-the-Loop Campaign Repository.” https://github.com/cmuchancel/SysMBench_Compiler_In_Loop (2026). Code, artifacts, and analysis scripts for this study. Accessed February 21, 2026.