Continuous AI Assistance for Model-Driven Engineering

Ludovic Apvrille[1] and Bastien Sultan[1]
[1]*LTCI, Télécom Paris, Institut Polytechnique de Paris, 450 route des Chappes, Sophia-Antipolis, France*
{*ludovic.apvrille, bastien.sultan*}*@telecom-paris.fr*

Keywords: MDE, MBSE, AI, LLM, Modeling suggestions, Continuous modeling, Incremental modeling

Abstract: Proactive AI-based assistants are now common in software engineering tools; however, few exist for Model-Driven Engineering (MDE) environments. Most existing AI assistants for MDE, particularly those based on large language models, require user interactions that can interrupt the modeling workflow. However, MDE is inherently a continuous process, involving successive cycles of diagram construction, verification, and modification. Relying on supplementary tools that require intensive interaction can therefore be time-consuming and disrupt engineers focus. Consequently, there is a need to shift AI-based modeling assistance paradigms to mechanisms that integrate naturally into the continuous MDE workflow. To address this need, the paper introduces ContinuousAI, a framework for AI-based continuous MDE assistance. Working alongside MDE engineers, ContinuousAI generates modeling suggestions either on demand or continuously, supporting the improvement of model quality throughout the engineering process. ContinuousAI has been implemented within the MDE toolkit TTool. Evaluation results show that ContinuousAI provides highly relevant suggestions while maintaining computation times and environmental footprints compatible with real-world continuous MDE usage.

1 INTRODUCTION

Artificial Intelligence-based assistance has become increasingly integrated into software tools, aiming to enhance user productivity. From early examples such as Microsoft Word's "Clippy" assistant, designed to suggest context-relevant actions, to modern features like real-time grammar and spell checking in text editors or predictive completion in chat and email applications, these forms of continuous assistance have progressively evolved to provide users with guidance as they work. Rather than requiring explicit user queries, such assistants continuously monitor user activity and accordingly offer support.

In recent years, similar forms of intelligent assistance have begun to emerge in software engineering environments. For instance, modern Integrated Development Environments (IDEs) now embed AI-driven tools capable of suggesting code completions, identifying design issues, or even generating test cases (Mäder et al., 2021; Finnie-Ansley et al., 2022). These continuous AI interactions intend to leave developers focus on higher-level problem-solving rather than syntactic or structural details.

In contrast, most current Model-Based Systems/Model-Driven Engineering (MBSE/MDE) environments, despite their advanced modeling and analysis capabilities, still rely on user-initiated interactions when leveraging AI (Di Rocco et al., 2025) or formal verification tools. Existing approaches, such as TTool-AI (Sultan and Apvrille, 2025), require the user to explicitly trigger the AI, select the relevant prompts, and specify the data to process. This workflow, while effective for targeted analyses, remains discontinuous and can interrupt the modeling flow.

This paper advocates for an emerging paradigm: **continuous AI assistance in MBSE**. In this approach, the AI operates continuously alongside the modeler, monitoring modeling activities to propose modeling relevant suggestions (e.g., missing aspects of a system specification). The proposed approach maintains the human-in-the-loop: the AI never modifies the model autonomously. Instead, it provides suggestions that the user can review, accept, refine, or discard. Furthermore, continuous AI assistance can integrate within modeling processes with formal verification, thus offering extra correctness in system design, and preventing both AI and human errors. The paper illustrates this AI-based assistance as a major extension of TTool-AI, and evaluates whether continuous suggestions are relevant.

The paper is organized as follows. Section 2

presents contextual elements on how AI has been integrated in TTool (TTool-AI) up to now. Then, Section 3 gives an overview of the related work, including how LLMs have already been coupled to MBSE, and if continuous aspects have already been considered for MBSE. Section 4 details the fundamental behind continuous AI. Section 5.1 shows how these fundamental has been practically implemented into TTool. An evaluation of the relevance of continuous AI in TTool is presented in Section 6. Finally, Section 7 concludes the paper.

2 TTOOL-AI: CONTEXT

This section provides a concise overview of the main features of TTool and TTool-AI, along with the current limitations of the latter.

2.1 TTool

TTool[1] is a free and open-source MDE toolkit (Apvrille et al., 2004) designed to support simulation and formal verification of UML models. Over time, it has been progressively extended with advanced modeling capabilities to better support design-space exploration through the DIPLODOCUS profile (Enrici et al., 2017), and with SysML modeling via the AVATAR profile (Apvrille et al., 2020). Currently, TTool provides support for safety, security, and performance analyses, both at the modeling and formal verification levels. It also provides interoperability features, including import and export capabilities for SysML v2 models, as well as a scripting mode that allows access to most of its features via a command-line interface. This enables, for example, loading a SysML v2 specification in textual form, running the internal model checker, and exporting the verification results.

2.2 TTool-AI

TTool-AI (Sultan and Apvrille, 2025) extends TTool with AI-assisted modeling and analysis capabilities. Its architecture is centered around an *AI Request Manager*, which interfaces with an AI engine (a Large Language Model (LLM)). The AI Request Manager handles three types of inputs: (1) the assistance feature selected by the user, (2) the system or model specification, and (3) any optional user prompt. These inputs are combined with feature-specific knowledge (e.g., the SysML syntax required for a particular diagram) to construct a sequence of prompts. This prompt sequence is then submitted to the AI engine, which produces a response tailored to the selected feature (such as generating a block diagram (Apvrille and Sultan, 2024), constructing an attack tree (Birchler De Allende et al., 2024a; Birchler De Allende et al., 2024b), analyzing consistency between two models (Sultan and Apvrille, 2024), etc.).

The user may directly apply the AI-generated proposals (for features involving model creation or modification) or iteratively refine them through additional interactions. In certain cases, a natural language processing phase precedes the AI analysis to better guide the model's "understanding" and ensure more accurate outcomes (Birchler De Allende et al., 2024a).

3 RELATED WORK

3.1 LLM-based Suggestions in Software Engineering

Continuous suggestion mechanisms for software engineers have long existed prior to the emergence of LLMs, with features such as auto-completion integrated into nearly all modern IDEs. Although auto-completion techniques have been proposed to support MDE (Mäder et al., 2021), they have predominantly targeted source code development.

The advent of LLMs has significantly broadened these capabilities, enabling their incorporation into a wide range of software engineering tools (Jin et al., 2024). Notable examples include LLM-based suggestion systems such as GitHub Copilot (Wermelinger, 2023), which relies on OpenAI Codex, a model specifically trained on software repositories to assist developers in source code generation and completion (Finnie-Ansley et al., 2022).

Nevertheless, while LLM-based continuous suggestion systems are now well established in source code development tools, only a limited number of studies have investigated their integration within MBSE/MDE tools.

3.2 LLMs for MBSE/MDE

As highlighted in the recent literature review by Di Rocco et al. (Di Rocco et al., 2025), the use of LLMs for MDE has become a dynamic research area over the past three years. Numerous contributions have targeted various MDE activities, including, among others, model generation from textual specifications.

[1] https://ttool.telecom-paris.fr

LLM-based model generation has indeed been evaluated for the design of different types of models, such as goal models (Chen et al., 2023), scenario-based models (Harel et al., 2024), UML class diagrams (Cámara et al., 2023), attack trees (Bircher De Allende et al., 2024a), and SysML models (Apvrille and Sultan, 2024; Sultan and Apvrille, 2024; Cibrián et al., 2025).

Our contribution builds upon TTool-AI, one of these LLM-based model generation approaches (Sultan and Apvrille, 2025). TTool-AI is an AI-powered modeling assistant integrated into the MDE toolkit TTool, as explained in the previous section. It relies on automated interactions with LLMs, enhanced through dynamic retrieval-augmented generation (RAG), which enriches the LLM with syntactic and semantic knowledge of the target modeling languages. In addition, it uses least-to-most prompting (Zhou et al., 2022) to decompose complex modeling tasks, and incorporates an automatic verification and feedback loop to iteratively refine the LLM's outputs until they meet the desired quality threshold.

Yet, an important limitation of the current version of TTool-AI lies in its interaction model: the user must explicitly trigger the AI each time assistance is desired, manually selecting the appropriate type of prompt and providing the relevant data (e.g., system specifications or model fragments). This manual process can be time-consuming and interrupts the natural modeling workflow, as it requires frequent context switching between model editing and AI interaction. Moreover, because the AI operates only upon explicit invocation, it lacks the ability to proactively anticipate user needs or offer context-aware suggestions during model construction.

The contribution introduced in this paper, that we call ContinuousAI, addresses these two limitations.

3.3 LLMs for continuous modeling suggestions

Bringing continuous modeling suggestions to users of MDE toolkits has been investigated in several previous works. Indeed, Di Rocco et al. (Di Rocco et al., 2025) listed model completion as a key MDE activity that can be effectively supported by LLMs. In particular, several studies have explored recommendation-based model completion, including those by Weyssow et al. (Weyssow et al., 2022) and Ben Chaaben et al. (Ben Chaaben et al., 2023).

The same authors also proposed MAGDA, a LLM-based continuous modeling assistant (Ben Chaaben et al., 2025), which focuses on the design of domain models using UML class diagrams. Due to its semantic focus, MAGDA provides three types of suggestions: adding classes, attributes (including names and types), and associations. Similarly to ContinuousAI, MAGDA injects the model into the AI assistant and relies on a predefined set of prompts. It supports three suggestion modes: on request, continuous, and at the end of model design, and these suggestions apply to the entire model.

By contrast, ContinuousAI additionally enables users to focus suggestions on a specific component and offers extra modes such as refreshing existing suggestions or refining a particular suggestion. Its scope also differs: ContinuousAI supports assistance for SysML block, state-machine, use-case, and requirement diagrams, as well as for attack trees. Moreover, because TTool integrates a mutation language (Sultan et al., 2025) relying on formally defined mutation operators that can translate suggestions into executable transformations, ContinuousAI can be combined with TTool's incremental model-checking algorithms (Coudert et al., 2024) for the supported models (currently block and state-machine diagrams). This combination is promising: continuous AI assistance inherently *is* incremental MDE. Pairing it with optimized incremental model checking can increase user confidence in applied suggestions while reducing the computational cost of repeated verifications.

4 AI ASSISTANCE FOR CONTINUOUS MDE: FUNDAMENTAL CONCEPTS

In essence, ContinuousAI is an AI-based framework that delivers continuous and proactive modeling suggestions to engineers throughout the entire model design cycle. As illustrated in Figure 1, it relies on a request manager that orchestrates an AI engine (e.g., a LLM) to generate suggestions based on contextual knowledge, the current model (in graphical or textual SysML/SysML v2 form), and a predefined set of prompts. The suggestion process operates both continuously (running as a background task) and on user request. The resulting suggestion sets can be applied, refined, saved, rejected, or updated, providing an interactive assistance mechanism that supports iterative model development. This section defines ContinuousAI using a semi-formal approach to better capture the concepts involved: definitions, how suggestions are computed, actions on suggestions, and finally the dynamic behavior behind the use of suggestions (including continuous aspects): this is our definition of

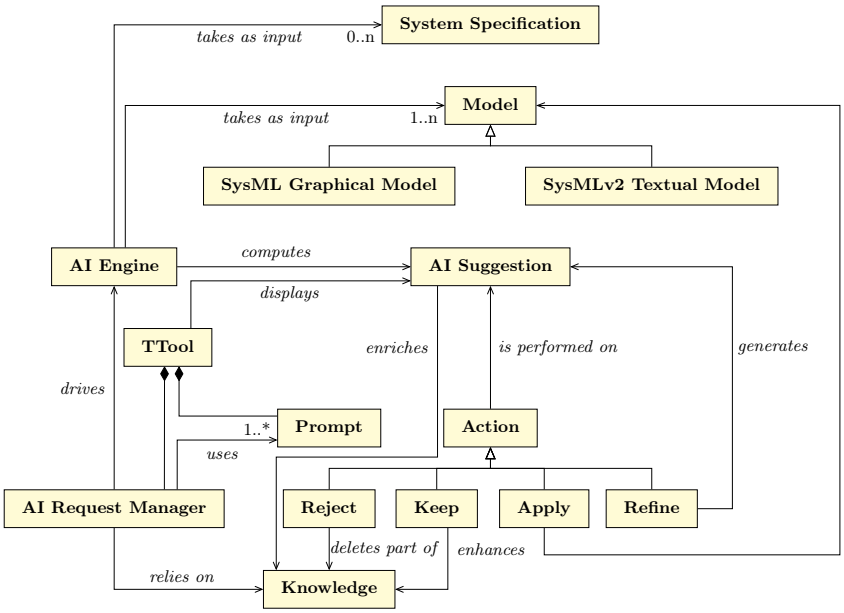

Figure 1: Concepts of ContinuousAI

"Continuous MDE".

4.1 Preliminary concepts

Definition 1 (Alphabet and models)**.**

1) We consider the following alphabet*:*

$$\mathcal{A} = \{a,A,\ldots,z,Z\} \cup \{0,\ldots,9\} \cup \{\ .\ ,\ \_\ ,\ ;\ ,\ ,\ ,\ \sqcup\}$$

The set of finite words induced by the Kleene closure over $\mathcal{A}$ is denoted with $\mathcal{A}*$, and the empty word with ε*.*

2) We denote with $\mathbb{M} = \mathbb{M}_{\mathsf{AV}} \sqcup \mathbb{M}_{\mathsf{v2}}$[2] the set of all models*. $\mathbb{M}_{\mathsf{AV}}$ contains models expressed in AVATAR semantics, and $\mathbb{M}_{\mathsf{v2}}$ in SysML v2 semantics.*

We consider in this paper that a model is composed of diagrams, referred to as *views*. We consider six types of views: block, state machine, use case, requirement and sequence diagrams, and also attack trees. Each view is composed of *elements*: for example, a block diagram contains *blocks* and *connections* between them, while a state machine diagram contains *states* and *transitions*. In addition, the model includes a relation that establishes associations between views and elements (for instance, linking a state machine diagram to the block it specifies within a block diagram). For a view v, we denote with $elements(v)$ the set of its elements.

[2] $A \sqcup B$ denotes the disjoint union of A and B, i.e. $A \cup B$ with $A \cap B = \varnothing$.

Definition 2 (Model)**.**

A model is a pair $m = (\mathbb{V}_m, \mathsf{associate})$, where:

- $\mathbb{V}_m = \{v_1,\ldots,v_n\}$ *is a finite set of* views*. We denote with $\mathbb{E}_m = \bigcup_{v \in \mathbb{V}_m} elements(v)$ the set of all elements contained in the views of m.*
- $\mathsf{associate} \subseteq (\mathbb{V}_m \cup \mathbb{E}_m)^2$ *is a relation that associates views and elements of m with other views and elements.*

We denote by $\mathbb{V}$ the universe set of all views, with $types = \{BD, SMD, UCD, REQ, SEQ, ATT\}$ the set of view types*, and with $\mathsf{type}_{\mathbb{V}} : \mathbb{V} \to types$ the function that associates each view with its type. For $type \in types$, $\mathbb{V}_{m_{type}} \triangleq \{v \in \mathbb{V}_m \mid \mathsf{type}_{\mathbb{V}}(v) = type\}$.*

We denote by $\mathbb{E}$ the universe set of all elements, and by $\mathbb{B} \subset \mathbb{E}$ the universe set of all blocks*.*

The set $\{\mathsf{type}_{\mathbb{V}}(v) \mid v \in \mathbb{V}_m\}$ is called the profile *of the model m. We denote it by $profile(m)$. $\forall m \in \mathbb{M}, profile(m) \cap \{BD, SMD\} \neq \varnothing \implies \mathsf{associate} \cap \big((\mathbb{E}_m \cap \mathbb{B}) \times \mathbb{V}_{m_{SMD}}\big)$ defines a bijection $\mathbb{E}_m \cap \mathbb{B} \xrightarrow{\sim} \mathbb{V}_{m_{SMD}}$.*

We denote with $m_\varnothing$ the empty model*, i.e. a model m such that $\mathbb{V}_m = \varnothing$.*

ContinuousAI involves (continuous) suggestions and modifications of the models. We thus define below model *mutation* functions to represent the modifications that can be brought to a model, either regarding the composition of its views, the number of its views or the associations between its views and elements. In the present paper, we consider only mu-

tations that preserve the semantic subset to which the model belongs (i.e., $\mathbb{M}_{\mathsf{AV}}$ and $\mathbb{M}_{\mathsf{v2}}$).

Definition 3 (Model mutation)**.**
A mutation *is a function*

$$\mu : \mathbb{M} \to \mathbb{M}$$

such that, for any model $m \in \mathbb{M}$*,* $\mu(m)$ *belongs to the same semantic subset as m, i.e.,* $m \in \mathbb{M}_{\mathsf{AV}} \implies \mu(m) \in \mathbb{M}_{\mathsf{AV}} \wedge m \in \mathbb{M}_{\mathsf{v2}} \implies \mu(m) \in \mathbb{M}_{\mathsf{v2}}$.

Given a model ⚓*,* $\lambda \in \mathbb{M}^{\mathbb{M}}$ *is called an* ⚓-anchor line *iff* $\mathbb{E}_{\text{⚓}} \cap \mathbb{E}_{\lambda(\text{⚓})} \neq \varnothing$.

4.2 Computing suggestions

ContinuousAI's core functionality consists of computing *suggestions* with the help of a generative AI agent (e.g., a LLM), based on a model, a specification, and contextual knowledge. This knowledge includes, for example, the syntax and semantics of AVATAR and SysML v2, as well as the expected structure of requested suggestions.

Definition 4 (Knowledge)**.**
The knowledge *is a set of finite words over the alphabet introduced in Definition 1.*

Definition 5 (Suggestion)**.**
A suggestion *is a finite word* $s \in \mathcal{A}*$ *that concatenates words describing: (1) the view and element targeted by the suggestion, (2) a description in natural language of the suggestion and (3) a description in natural language of how to modify the model to take the suggestion into account.*

In the context of ContinuousAI, some of these suggestions may be flagged as *kept*, and related to a *parent suggestion*. They are stored in a structure of suggestions.

Definition 6 (Structure of suggestions)**.**
A structure of suggestions *is a triplet* $(suggestions_{kept}, suggestions_{produced}, \mathcal{R})$ *such that:*

- $suggestions_{kept}$ *and* $suggestions_{produced}$ *are two disjoint sets of* suggestions*.*
- $\mathcal{R} \subseteq (suggestions_{kept} \sqcup suggestions_{produced})^2$ *is such that* $\mathcal{R}^{-1}$ *is a function and its transitive closure* $\mathcal{R}^{+}$ *is a noetherian order.*

$\mathbb{S}$ *denotes the set of all* structures of suggestions*.*

Given a model m, some *knowledge* and a $specification$, the first step of ContinuousAI is to compute a set of suggestions that improves m with respect to the specification (i.e., making m better satisfy the specification), the *knowledge*, and the LLM's own knowledge base (e.g., making m better respect best modeling practices). ContinuousAI also relies on pre-defined prompts to guide the LLM in computing the suggestion. Because of the stochastic nature of LLMs, several sets of suggestions are possible in response to a same input $(model, specification, knowledge, prompts)$. We therefore model suggestion computation as a function that returns a set of possible responses, from which one suggestion is chosen arbitrarily:

Definition 7 (Suggestion computation)**.**
The suggestion computation *is a function*

$$\begin{array}{rll} compute: & \mathbb{M} \times \wp(\mathcal{A}*) & \longrightarrow \wp(\wp(\mathcal{A}*)) \\ & (m, w) & \longmapsto suggestions \end{array}$$

where $w \supseteq \{specification\} \cup knowledge \cup prompts$ *and* $(m \neq m_{\varnothing}) \vee (specification \neq \varepsilon)$.

A suggestion set *is any element chosen arbitrarily from the output set of compute, and a* suggestion *is any element of this set.*

4.3 Actions on suggestions

Once a suggestion has been generated, ContinuousAI offers several types of *actions* that can be performed based on the suggestion. Those actions are:

- *keep*: Within a structure of suggestions, the suggestion is added to the set of kept suggestions and deleted from the set of produced suggestions.
- *reject*: Within a structure of suggestions, the suggestion is deleted from the set of produced suggestions.
- *refine*: ContinuousAI computes a new suggestion set that refine the original suggestion with additional details.
- *apply*: The model is modified according to the suggestion. More precisely, ContinuousAI first tasks the LLM with computing a mutation representing the transformation of the model recommended by the suggestion, and then applies this mutation to the model.

Definition 8 (Actions)**.**
For a given suggestion:
1) The keep *action is a function:*

$$\begin{array}{l} keep_{suggestion} : \mathbb{S} \longrightarrow \mathbb{S} \\ (S_{kept}, S_{prod}, \mathcal{R}) \longmapsto (S_{kept} \cup \{suggestion\}, \\ \qquad\qquad S_{prod} \setminus \{suggestion\}, \mathcal{R}) \end{array}$$

2) The reject *action is a function*

$$\begin{array}{l} reject_{suggestion} : \mathbb{S} \longrightarrow \mathbb{S} \\ (S_{kept}, S_{prod}, \mathcal{R}) \longmapsto (S_{kept}, S_{prod} \setminus \{suggestion\}, \mathcal{R}') \end{array}$$

where $\mathcal{R}' = \mathcal{R} \cap (S_{kept} \sqcup S_{prod} \setminus \{suggestion\})^2$.

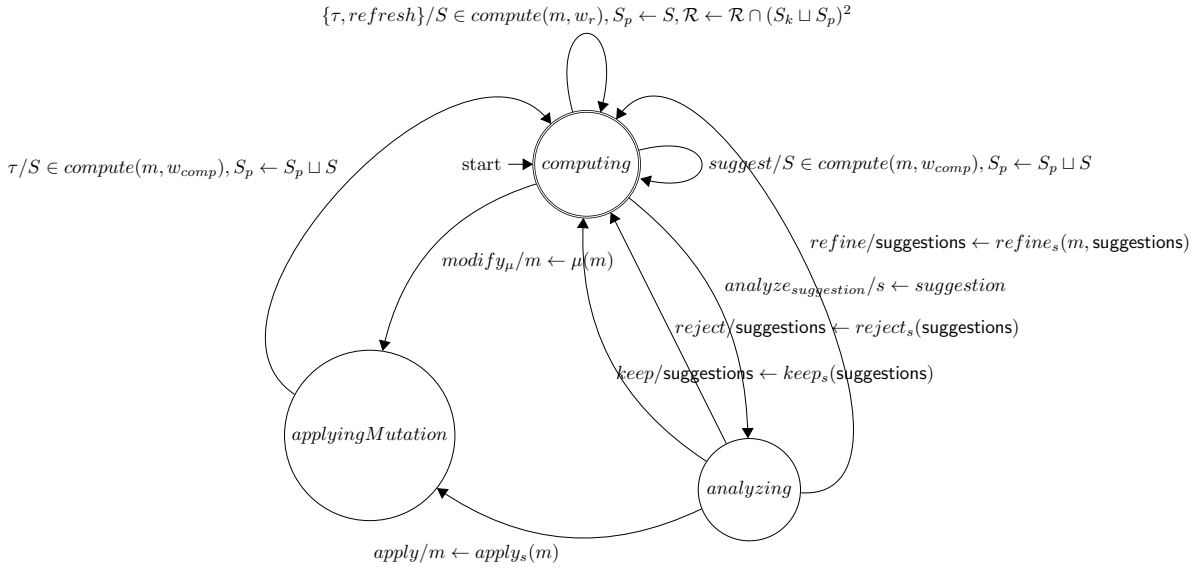

Figure 2: Mealy machine modeling ContinuousAI dynamic behavior

3) The refine *action is a function:*

$$refine_{suggestion} : \mathbb{M} \times \mathbb{S} \longrightarrow \mathbb{S}$$
$$\big(m, (S_{kept}, S_{prod}, \mathcal{R})\big) \longmapsto (S_{kept}, S_{prod} \sqcup S, \mathcal{R}')$$

where:

- $S \in compute(m, w)$ *s.t.* $S \cap (S_{kept} \cup S_{prod}) = \varnothing$
- $suggestion \in w$
- *w contains prompts tasking with the refinement of* $suggestion$
- $\mathcal{R}' = \mathcal{R} \cup \{(suggestion, s) \mid s \in S\}$.

4) The apply *action is a function*

$$apply_{suggestion} : \mathbb{M} \longrightarrow \mathbb{M}$$
$$m \longmapsto \mu(m)$$

where μ *is a function chosen arbitrarily in* $compute_\mu(m, suggestion)$*, with*

$$compute_\mu : \mathbb{M} \times \mathcal{A}* \longrightarrow \wp(\mathbb{M}^{\mathbb{M}})$$
$$m, suggestion \longmapsto mutations$$

such that $mutations$ *is a set of possible mutations representing the transformation of m corresponding to* $suggestion$*. They are* $m-$*anchor lines iff* $m \neq m_\varnothing$.

4.4 Continuous dynamic behavior

The continuous dynamic behavior of ContinuousAI is defined by the Mealy machine (extended with a memory $s \in \mathcal{A}*$) shown in Figure 2. This Mealy machine operates on an alphabet Σ containing the following input symbols, each representing a user action:

- *suggest*: Requests the computation of suggestions.
- *refresh*: Requests the update of unkept suggestions.
- $modify_\mu$: Applies a mutation μ to the model.
- $analyze_{suggestion}$: Selects a suggestion for analysis.
- *refine*: Requests the refinement of the currently analyzed suggestion.
- *apply*: Requests the application of the currently analyzed suggestion.
- *keep*: Requests keeping the currently analyzed suggestion.
- *reject*: Requests the rejection of the currently analyzed suggestion.

We also consider an additional symbol τ, which represents the absence of user action. τ-transitions are thus triggered spontaneously by the automaton whenever no external event from Σ occurs, meaning that the automaton evolves spontaneously (and thus continuously) in the absence of input.

For the output symbols, $m \in \mathbb{M}$ denotes the model under design, $\text{suggestions} = (S_k, S_p, \mathcal{R})$ is a structure of suggestions, and:

- $w_{comp} \in \wp(\mathcal{A}*)$ contains prompts instructing the LLM to generate suggestions.
- $w_r \in \wp(\mathcal{A}*)$ contains prompts instructing the LLM to reevaluate the suggestions in S_p.
- $s \in \mathcal{A}*$ is the suggestion under analysis.

- $keep_s$, $reject_s$, $refine_s$ and $apply_s$ are the functions defined in Definition 8.
- $S \in compute(m,w)$ is chosen such that $S \cap (S_k \cup S_p) = \varnothing$.

This continuous dynamic behavior enables ContinuousAI to compute suggestions at the user's request (e.g., $computing \xrightarrow[S \in compute(m,w_s), S_p \leftarrow S_p \sqcup S]{suggest} computing$), and, in the absence of user input, to continuously re-evaluate previously computed suggestions ($computing \xrightarrow[S \in compute(m,w_r), S_p \leftarrow S, \mathcal{R} \leftarrow \mathcal{R} \cap (S_k \sqcup S_p)^2]{\{\tau, refresh\}} computing$).

5 IMPLEMENTATION OF ContinuousAI IN TTool

This section describes how ContinuousAI has been integrated into the MBSE toolkit TTool, covering both graphical modeling and textual (SysML v2) modeling. All concepts given in Figure 1 have been implemented in TTool[3].

5.1 Graphical Modeling with Continuous AI

5.1.1 Suggestion generation (*compute*)

As formalized in Definition 5, a suggestion is composed of several elements:

- A title
- A description indicating whether the suggestion concerns the structural aspects, behavioral aspects, or both, and an indication of the specific model element affected by the suggestion (if any)
- A textual explanation detailing the content of the suggestion
- Guidance on how the model can be manually modified to implement the suggested change.

Our implementation follows the Mealy machine illustrated in Figure 2. Consequently, TTool-AI generates overall modeling suggestions either automatically upon model updates ($modify_\mu$) or when the user explicitly requests a new list of suggestions (*suggest*). Suggestions targeting a specific model element can also be produced. In this case, the *suggest* trigger is either an explicit user action (a click) or an implicit one, occurring when the user hovers the mouse pointer over the element for a few seconds. In our implementation of *compute* (see Definition 7), ContinuousAI returns a set of (at most) ten suggestions when targeting the entire model, or a single suggestion when targeting a model element. Our implementation also enables unkept suggestions to be continuously refreshed in the background, without requiring any user input (τ-transitions of the Mealy machine). Additionally, our implementation of *compute* incorporates a feedback loop that ensures the AI-generated suggestions adhere to the required format constraints. This loop iteratively interacts with the LLM until a correctly structured suggestion is produced, or until a maximum iteration limit is reached. ContinuousAI accesses the LLM by querying its chat completion API.

5.1.2 Suggestion handling (actions)

Once suggestions have been computed, a user can perform the actions introduced in Definition 8:

- *Keep* the suggestion (default action). The suggestion remains in the *structure of suggestions*, displayed either as a bubble attached to the model element or in the project navigation panel of TTool.
- *Reject* the suggestion. The suggestion is removed from both the list and the GUI.
- *Refine* the suggestion. The suggestion is split into a set of sub-suggestions. This is particularly useful when the suggestion is high-level and thus cannot be directly applied to the model. For instance, a suggestion like: "Reduce the number of connections by better exploiting modular aspects of blocks" could be refined in "move this function to this block", "remove that signal from that block", etc.
- *Apply* the suggestion. This feature is not yet automated for all diagram types, however, the necessary tools already exist in TTool to complete its automation for models in $\{m \in \mathbb{M}_{\mathsf{AV}} \mid profile(m) \supseteq \{BD, SMD\}$. Currently, it involves tasking the AI Engine with translating the selected suggestion into a sequence of AMULET mutation commands, using a pre-existing TTool-AI feature (Sultan and Apvrille, 2025). AMULET is TTool's mutation language, which enables the scripting of all modifications applied to block and state machine diagrams (Sultan et al., 2025). Once the AMULET script is generated, it is executed on the model by TTool's internal AMULET compiler and the model is modified.

[3]The source code of our implementation, including the predefined prompts we use, is available at: https://gitlab.telecom-paris.fr/mbe-tools/TTool/-/tree/master/src/main/java/ai.

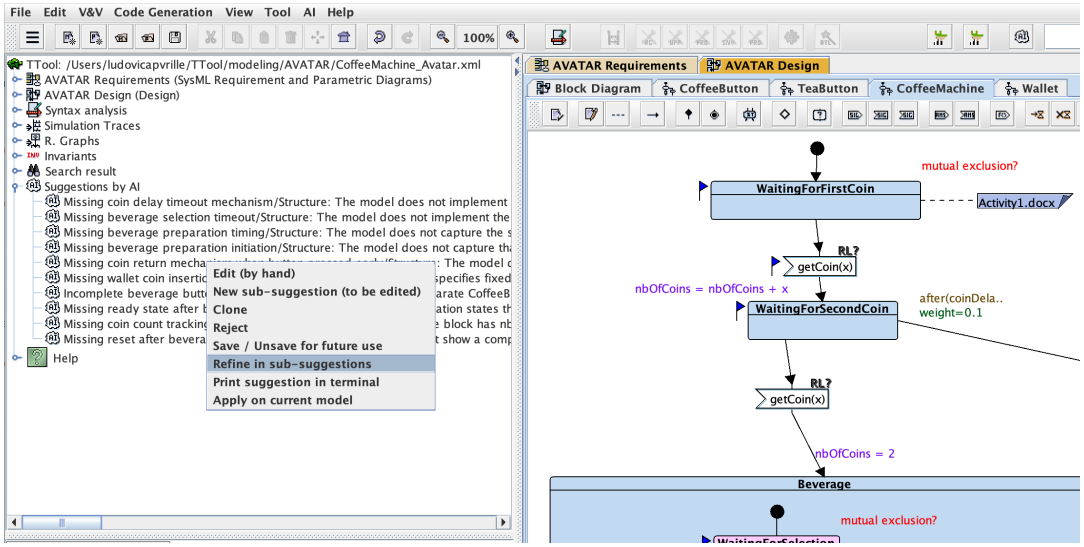

Figure 3: A list of suggestions generated by ContinuousAI for an AVATAR design model

Figure 3 illustrates a list of suggestions generated for an AVATAR design model (a block diagram and a set of state machine diagrams). Different actions can be performed on each suggestion, and suggestions can also be manually edited if needed. Figure 4 presents an example of a suggestion that applies to a specific element. Each time the model is modified, the global list of suggestions is automatically updated (ContinuousAI). In both cases, the examples were produced using the AVATAR Coffee Machine model available in the TTool repository.

5.2 SysML v2 Textual Modeling with Continuous AI

As discussed earlier, TTool is scriptable, including its AI functions. For instance, the following script can be used to load a SysML v2 model from its textual form, and then to generate suggestions.

Listing 1: Script for suggestions

```
action avatar-load-sysmlv2 machine.
    sysmlv2
ai setAIModel openai/gpt-oss-20b
ai setSystemSpecification This coffee
    machine dispenses a beverage ...
ai giveSuggestions 5 // at most 5
    suggestions
```

The typical feedback may be as follows:

Listing 2: Result of script

```
5 suggestions:
        FirstCoinTimeout/Structure: The
            system specification
            requires that if more than
            30s elapse between the
            first and second coin, the
            machine must ...
        SecondCoinTimeout/Structure:
            After two coins are
            deposited, if no beverage
            button is pressed...
    ...
```

6 CONTINUOUS ASSISTANCE IN PRACTICE

This section presents an evaluation conducted to assess the practical relevance of the ContinuousAI concepts and their implementation in TTool. It aims to provide basic benchmarks for understanding the potential impact of ContinuousAI within a MDE process.

6.1 Evaluation Setting

We evaluated our implementation of ContinuousAI according to four main criteria: (1) the relevance of its suggestions, (2) their ability to improve the targeted model (or view element), (3) the average time required to compute a set of suggestions (i.e., the mean execution time of our implementation of the *compute* function) and (4) the environmental impact of ContinuousAI (in terms of power consumption and amount of CO_2 emitted per suggestion).

To collect the data needed to assess these crite-

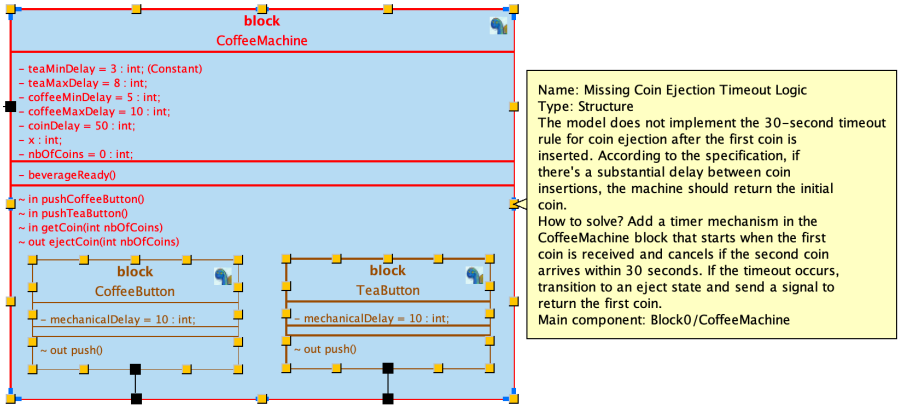

Figure 4: ContinuousAI generates a suggestion specific to a SysML block of a block diagram

ria, we conducted the following evaluation procedure. Each author of this paper constructed a set of SysML views comprising three requirement diagrams (RD), three use case diagrams (UCD), and three design models (each consisting of a block diagram and a corresponding set of state machine diagrams). These diagrams were extracted from exams and labs produced by final-year engineering students and corresponded to three distinct system specifications: a packaging chain control system, a space-based system, and a smart greenhouse control system. The diagrams were selected such that they reflect heterogeneous quality levels (for instance, the second set of views included RDs that had been graded 5/10, 6.7/10, and 10/10, respectively). The diagrams used for this evaluation were of moderate complexity: for instance, RDs contained up to 24 requirements, and UCDs included up to 6 actors and 10 use cases.

Subsequently, each author performed the following actions for every diagram in their respective sets:

1. Requested ContinuousAI to generate suggestions for the entire diagram using two different locally executed LLMs[4]: GPT-OSS-20b and Qwen3-Coder:30b.
2. Randomly selected one element within each view and requested ContinuousAI to generate a suggestion specifically for that element, again using both LLMs.
3. Recorded the computation times reported by our ContinuousAI implementation.
4. Rated the suggestions based on their relevance.
5. Assessed the potential evolution of the model if the suggestions were applied, using a seven-level qualitative scale: degradation (-3 to -1), neutral (0), and improvement (+1 to +3).

6.2 Results

Table 1 summarizes the results of our evaluation. The reported values are averages per diagram type (as each of the two evaluation sets included three diagrams of each kind). Note that the complete result tables, the suggestions computed by ContinuousAI, detailed instructions on how to use ContinuousAI, and an example base model (including its suggestions) are available in a public GitHub repository[5].

Overall, we first note that although there is some inter-evaluator variability in the scores assigned to the relevance of the suggestions and to the qualitative assessment of model improvement, the overall trends are consistent across both evaluation series: suggestions generated by ContinuousAI + Qwen3 are better than those generated by ContinuousAI + GPT-OSS, and the application of the suggestions generally leads to improved model quality. It is also important to keep in mind that the LLMs used in this evaluation were locally hosted models, which may have lower performance than provider-hosted models such as GPT-5. However, locally hosted models likely better reflect real-world industrial scenarios, where confidentiality constraints often limit the use of externally hosted (AI) services.

These results can be interpreted with respect to our four evaluation criteria as follows:

(1) Relevance of the suggestions

[4]The computer used for the evaluation had the following configuration: an Intel Core Ultra 9 285 processor (24 cores with Hyper-Threading, 2.5–5.6 GHz Turbo, 36 MB cache, 65 W TDP), an NVIDIA RTX 5000 Ada GPU with 32 GB of VRAM (12 800 CUDA cores, 400 fourth-generation Tensor cores, and 100 third-generation RT cores, 4× DisplayPort outputs), and 64 GB of DDR5-5600 MT/s memory.

[5]https://github.com/ZebreDeSoixanteQuatorzeCanons/ContinuousAI

Table 1: Evaluation results

		Before applying suggestions					*After applying suggestions*		
		Quality of suggestions: score /10					**Model quality evolution (-3 to +3)**		
		GPT-OSS-20b			**Qwen3**				
	Model grade /10		**Score**	**Time (in ms)**	**Score**	**Time (in ms)**		**Score OSS**	**Score Qwen**
RD	9.4		7.7	9,337	8.7	17,241		+1.7	+2.5
UCD	8.9		9	6,544	9	26,088		+2.7	+2.7
BD	9.2		8.8	8,070	8.7	15,644		+2.7	+2.7
		Average	8.3	7,940	8.8	21,664	**Average**	+2.2	+2.6

(1a) Suggestions bearing on the whole diagram

		Before applying suggestions					*After applying suggestions*		
		Quality of suggestions: score /10					**Model quality evolution (-3 to +3)**		
		GPT-OSS-20b			**Qwen3**				
	Model grade /10		**Score**	**Time (in ms)**	**Score**	**Time (in ms)**		**Score OSS**	**Score Qwen**
RD	9.4		8.7	1,935	5.7	9,564		+2.7	+1.7
UCD	8.9		6.3	1,995	10	8,878		+1.7	+3
BD	9.2		9.2	2,571	10	11,885		+2.7	+3
		Average	7.5	1,965	7.8	9,221	**Average**	+2.2	+2.3

(1b) Suggestions bearing on a given component

		Before applying suggestions					*After applying suggestions*		
		Quality of suggestions: score /10					**Model quality evolution (-3 to +3)**		
		GPT-OSS-20b			**Qwen3**				
	Model grade /10		**Score**	**Time (in ms)**	**Score**	**Time (in ms)**		**Score OSS**	**Score Qwen**
RD	7.2		5.7	11,532	7.3	36,205		+1.3	+2.3
UCD	8.3		5.5	9,731	7	31,132		+1	+1.3
BD	6.7		4.5	9,420	7.3	26,324		+1.3	+2.3
		Average	5.2	10,228	7.2	31,220	**Average**	+1.2	+2

(2a) Suggestions bearing on the whole diagram

		Before applying suggestions					*After applying suggestions*		
		Quality of suggestions: score /10					**Model quality evolution (-3 to +3)**		
		GPT-OSS-20b			**Qwen3**				
	Model grade /10		**Score**	**Time (in ms)**	**Score**	**Time (in ms)**		**Score OSS**	**Score Qwen**
RD	7.2		6.7	5,180	10	5,957		+1.7	+2
UCD	8.3		6.7	5,559	6.7	8,788		+2	+1.3
BD	6.7		6.7	5,991	3.3	3,882		+1.7	+0.7
		Average	6.7	5,578	6.7	6,209	**Average**	+1.8	+1.3

(2b) Suggestions bearing on a given component

Table 2: Average energy consumption and CO2 emissions per request and suggestion

	GPT-OSS-20b	**Qwen3**
Average power consumption per request (J)	966	1529
Grams of CO2 per request	0.0135	0.0214
Average power consumption per suggestion (J)	293	463
Grams of CO2 per suggestion	0.0041	0.0065

The suggestions are, overall, relevant to highly relevant. In nearly all cases, more than half of the proposed suggestions were judged relevant by the evaluators, and in 75% of cases, more than two thirds were considered relevant. On average, suggestions produced using Qwen3 as the underlying LLM were rated higher than those generated using GPT-OSS.

(2) Ability to improve model quality

Across our whole evaluation (covering 72 analysis cases), the application of the suggestions would have led to a diagram degradation in only one case, no change in diagram quality in seven cases, and an improvement in 64 cases. Consequently, in 71 out of 72 cases (98.6%), using ContinuousAI contributed to maintaining or improving model quality, and in 64 out of 72 cases (88.9%), it resulted in an improvement.

(3) Average computation time required for a set of suggestions

This aspect is crucial for assessing the practical applicability of ContinuousAI in real MDE contexts and is measured through objective metrics. The computation times are satisfactory: for full-diagram suggestions, the average computation time was 9 seconds with ContinuousAI + GPT-OSS and 26.4 seconds with ContinuousAI + Qwen3. For single-element suggestions, the averages were 3.8 seconds and 7.7 seconds, respectively. In our opinion, these response times are fully compatible with continuous assistance: according to our experimentation, on average, depending on the underlying LLM, the list of suggestions is then returned to the user 9 to 27 seconds after

their last interaction with the modeling environment. Again, these measurements correspond to locally executed LLMs: performance may differ when using remotely hosted models provided by LLM vendors.

(4) Power consumption and CO_2 emissions Table 2 summarizes the average power consumption and CO_2 emissions per suggestion computed on our local LLM server across the first evaluation. Note that our CO_2 estimates are based on the kWh-to-CO_2 conversion factors provided by RTE, the French electricity transmission system operator in October 2025; these values may vary depending on the national energy mix of each country. The results appear reasonable: on average, each request emits $38\times$ less CO_2 than an average Google query (0.2 g CO_2 (Google, 2009)). This footprint is, however, expected to be higher when using ContinuousAI with remotely hosted models provided by LLM vendors, as existing (albeit informal and approximate) estimates suggest substantially greater CO_2 emissions per ChatGPT query (Tomlinson et al., 2024).

6.3 Threats to validity

This evaluation provides a basis for discussing the practical relevance of ContinuousAI. However, it has several limitations that must be taken into account to properly understand its scope.

First, as stated earlier, the assessment of both (1) the quality of the initial models and (2) the relevance of the generated suggestions is qualitative and therefore inherently subjective. Some inter-evaluator variability was observed. Nevertheless, consistent trends emerged across the two evaluation rounds.

In addition, we did not evaluate the actual application of the suggestions. Instead, we assessed their relevance and the extent to which they were expected to improve the models if applied. In particular, we did not (1) use TTool-AI to generate mutation scripts for modifying BDs based on the suggestions, nor (2) modify RDs and UCDs according to the proposed changes.

Related to this limitation, at the time of this evaluation, the application of a suggestion s through the function $apply_s$ is supported in TTool only for SMDs and BDs, via the mutation mechanisms provided by TTool-AI. For other diagram types, no formal mutation language is currently available. Consequently, users must manually apply the suggestions for these diagrams.

Finally, the base models used in the evaluation are of moderate complexity, with at most 24 requirements for RDs and at most 6 actors and 10 use cases for UCDs.

7 CONCLUSION

The paper introduces ContinuousAI, a framework for AI-based continuous MDE assistance. Fundamentally, ContinuousAI is driven by a Mealy machine that triggers the computation of modeling suggestions, stores them in a structured set, and performs various actions on these suggestions (refinement, application, rejection, etc.) based on user decisions that remain central to the decision loop. This Mealy machine operates continuously within the MDE process, supporting proactive, continuous background computation of suggestions. Furthermore, ContinuousAI's conceptual reliance on model mutations for suggestion integration ensures compatibility with incremental verification approaches. Practically, ContinuousAI has been implemented within TTool, applying it to SysML (including SysML v2) models. Similar to TTool-AI, ContinuousAI relies on RAG, automated feedback loops, and predefined prompt sets to interact in the background with the underlying LLMs. From a usage paradigm perspective, ContinuousAI brings a major improvement to TTool-AI, substantially reducing user interaction effort while keeping the user fully in control through simple, intuitive actions (such as single-click interactions within the TTool model design window).

The paper evaluates the relevance of ContinuousAI through practical experiments on various SysML diagrams (requirements, use case, and block diagrams). Overall, the evaluation demonstrates significant benefits from using ContinuousAI's suggestions, improving or maintaining diagram quality in 98.6% of the cases according to our criteria. It also shows reasonable suggestion computation times, even when using locally run LLMs, making the framework practically compatible with continuous MDE. Furthermore, environmental impact measurements indicate that such continuous AI assistance has a very acceptable footprint.

Future work will focus on extending the evaluation of ContinuousAI in real-world scenarios, comparing the quality of models produced by engineers using ContinuousAI versus those using TTool without it. Additional developments will include fully automating the *apply* function for all supported diagram types and integrating ContinuousAI with incremental model-checking approaches, to open the way to an AI-assisted, continuous, and verified MDE.

REFERENCES

Apvrille, L., Courtiat, J.-P., Lohr, C., and de Saqui-Sannes, P. (2004). TURTLE: a real-time UML profile supported by a formal validation toolkit. *IEEE Transactions on Software Engineering*, 30(7):473–487.

Apvrille, L., de Saqui-Sannes, P., and Vingerhoeds, R. (2020). An Educational Case Study of Using SysML and TTool for Unmanned Aerial Vehicles Design. *IEEE Journal on Miniaturization for Air and Space Systems*, 1(2):117–129.

Apvrille, L. and Sultan, B. (2024). System Architects are not Alone Anymore: Automatic System Modeling with AI. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD'24*. INSTICC.

Ben Chaaben, M., Burgueño, L., David, I., and Sahraoui, H. (2025). On the Utility of Domain Modeling Assistance with Large Language Models. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Ben Chaaben, M., Burgueño, L., and Sahraoui, H. (2023). Towards using few-shot prompt learning for automating model completion. In *2023 IEEE/ACM 45th international conference on software engineering: New ideas and emerging results (ICSE-NIER)*, pages 7–12. IEEE.

Birchler De Allende, A., Sultan, B., and Apvrille, L. (2024a). Automated Attack Tree Generation Using Artificial Intelligence and Natural Language Processing. In *International Conference on Risks and Security of Internet and Systems*, pages 141–156. Springer.

Birchler De Allende, A., Sultan, B., and Apvrille, L. (2024b). From Attack Trees to Attack-Defense Trees with Generative AI & Natural Language Processing. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 561–569, New York, NY, USA. Association for Computing Machinery.

Cámara, J., Troya, J., Burgueño, L., and Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling*, pages 1–13.

Chen, B., Chen, K., Hassani, S., Yang, Y., Amyot, D., Lessard, L., Mussbacher, G., Sabetzadeh, M., and Varró, D. (2023). On the Use of GPT-4 for Creating Goal Models: An Exploratory Study. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 262–271. IEEE.

Cibrián, E., Olivert-Iserte, J., Llorens, J., and Álvarez Rodríguez, J. M. (2025). An agent-based approach for the automatic generation of valid sysmlv2 models in industrial contexts. *Computers in Industry*, 172:104350.

Coudert, S., Apvrille, L., Sultan, B., Hotescu, O., and de Saqui-Sannes, P. (2024). Incremental and Formal Verification of SysML Models. *SN Comput. Sci.*, 5(6).

Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P. T., and Rubei, R. (2025). On the use of large language models in model-driven engineering. *Softw. Syst. Model.*, 24(3):923–948.

Enrici, A., Apvrille, L., and Pacalet, R. (2017). A Model-Driven Engineering Methodology to Design Parallel and Distributed Embedded Systems. *ACM Trans. Des. Autom. Electron. Syst.*, 22(2):34:1–34:25.

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian computing education conference*, pages 10–19.

Google (2009). Powering a Google search. https://googleblog.blogspot.com/2009/01/powering-google-search.html.

Harel, D., Katz, G., Marron, A., and Szekely, S. (2024). On Augmenting Scenario-Based Modeling with Generative AI. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD'24*. INSTICC.

Jin, H., Huang, L., Cai, H., Yan, J., Li, B., and Chen, H. (2024). From LLMs to LLM-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479*.

Mäder, P., Kuschke, T., and Janke, M. (2021). Reactive Auto-Completion of Modeling Activities. *IEEE Transactions on Software Engineering*, 47(7):1431–1451.

Sultan, B. and Apvrille, L. (2024). AI-Driven Consistency of SysML Diagrams. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS '24, page 149–159, New York, NY, USA. Association for Computing Machinery.

Sultan, B. and Apvrille, L. (2025). TTool-AI: A Large Language Model-Based Assistant for Model Driven Engineering. *SN Comput. Sci.*, 6(7).

Sultan, B., Frénot, L., Apvrille, L., Jaillon, P., and Coudert, S. (2025). AMULET: A Mutation Language Enabling Automatic Enrichment of SysML Models. *ACM Trans. Embed. Comput. Syst.*, 24(3).

Tomlinson, B., Black, R. W., Patterson, D. J., and Torrance, A. W. (2024). The carbon emissions of writing and illustrating are lower for AI than for humans. *Scientific Reports*, 14(1):3732.

Wermelinger, M. (2023). Using github copilot to solve simple programming problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 172–178.

Weyssow, M., Sahraoui, H., and Syriani, E. (2022). Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling*, 21(3):1071–1089.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., et al. (2022). Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.