Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

LLM ASSISTED VALUE STREAM MAPPING

Micha Selak[1], Dirk Krechel[1], Adrian Ulges[1], Sven Spieckermann[2], Niklas Stoehr[2], and Andreas Loehr[2]

[1]Dept. of DCSM, RheinMain University of Applied Sciences, Wiesbaden, HESSE, GERMANY
[2]SimPlan AG, Hanau, HESSE, GERMANY

ABSTRACT

The correct design of digital value stream models is an intricate task, which can be challenging especially for untrained or inexperienced users. We address the question whether large language models can be adapted to "understand" value stream's structure and act as modeling assistants, which could support users with repairing errors and adding or configuring process steps in order to create valid value stream maps that can be simulated. Specifically, we propose a domain-specific multi-task training process, in which an instruction-tuned large language model is fine-tuned to yield specific information on its input value stream or to fix scripted modeling errors. The resulting model – which we coin Llama-VaStNet – can manipulate value stream structures given user requests in natural language. We demonstrate experimentally that Llama-VaStNet outperforms its domain-agnostic vanilla counterpart, i.e. it is 19% more likely to produce correct individual manipulations.

1 INTRODUCTION

Value streams model the production and delivery processes of goods or services. Their digital representations, referred to as value stream *maps* (VS)s or value stream *models*, typically illustrate the sequence of process steps, information flows, material movements, and lead times involved. This provides a holistic view of value creation from start to finish. VSs allow experts to visualize, simulate, analyze and optimize the underlying processes. Since value stream *mapping* (VSM) is considered a key methodology in lean management (Rother and Shook 2003), it is essential to enable users to create and manipulate their digital VSs effectively. Furthermore, in order to be able to simulate the dynamics of stocks and process utilization's of a value stream in a discrete-event simulation model, the corresponding value stream map must be syntactically correct and semantically sound.

However, users often perceive value stream digitization as challenging. Structural or logical errors are frequently made, leading to VSs unsuitable for the execution in a simulation engine. This is rooted in the tendency of the underlying models to become complex and cluttered with information: Formally, VSs are attributed graphs which consist of interconnected nodes representing the underlying process's steps. Both nodes and edges come in different types with various type-specific attributes. The plethora of modeling options renders the task of VSM difficult for non-experts.

We argue that the challenge of value stream digitization can be compared with the task of *software programming*: In both cases, developers need to understand the underlying syntax and interfaces, and in both cases, finding "bugs" such as missing connections or out-of-date attributes is an intricate task, since failures in simulation (or in program executions, respectively) often result in cryptic error messages.

This raises the question whether recent improvements in software development by artificial intelligence – specifically, large language models (LLMs) – could be adopted for VSM. In software development, LLM-based chat assistants such as GitHub Copilot or Cursor have become wide-spread, supporting the user with code generation and other everyday programming tasks (Hou et al. 2024). LLMs have also already been used to automate the creation of simulation models of logistical systems in the form of Python code (Jackson et al. 2023).

This paper aims to develop an LLM-based assistant capable of manipulating and correcting VSs based on user instructions, as shown in Figure 1: The user expresses a need to fix or improve his current VS as a question, the LLM interprets the request together with the parsed VS, and answers with a modification to the VS specified in a function call. This modification is then displayed to the user in the UI.

To build such an assistant, one has to address the *low-exposure problem*: LLMs' capability of "understanding" a formal language is known to correlate strongly with the frequency with which the language was encountered during training. This phenomenon is well-known for programming languages: Programming assistants handle Common languages such as Python – which can be found in vast amounts of open-source code – well, but struggle with infrequent, *low-resource* languages such as racket (Sathvik et al. 2024).

Since VSM is a low-resource problem field, vanilla LLMs cannot be expected to "understand" VSs properly. To overcome this problem, we suggest a self-supervised fine-tuning, in which we expose the pre-trained LLM to VSs, enhancing the model's understanding of their domain-inherent concepts and familiarizing it with the specific format of the VSs used. During this fine-tuning, the model receives a VS along with a user instruction and is primed to output required modifications to the VS. We have designed nine types of different training questions (or *tasks*), ranging from simple ones such as identifying a node by its name, to complex ones such as adding a new node with specific connections. We combine these tasks in a multi-task training, and coin the resulting model *Llama-VastNet* (for **Va***lue* **St***ream* **Net***work*). Finally, we demonstrate experimentally that our fine-tuning improves the model's capability to conduct VS modifications. We hope that our approach lowers the barrier to simulate value streams and enables users to utilize VSM more effectively, thereby improving the accessibility for a broader audience.

To summarize, our contributions are:

- We present Llama-VastNet, an LLM that – given user request – suggests repairs and other modifications to VSs. To our knowledge, Llama-VastNet is the first attempt to adapt an LLM for the specific domain of VSM.
- Llama-VastNet is based on a novel fine-tuning procedure to improve the understanding of VSs as a low-resource language. The procedure is based on nine VSM-specific training tasks.
- In quantitative experiments on training tasks as well as on real user instructions, we demonstrate that our fine-tuning significantly improves the models ability to modify VSs.

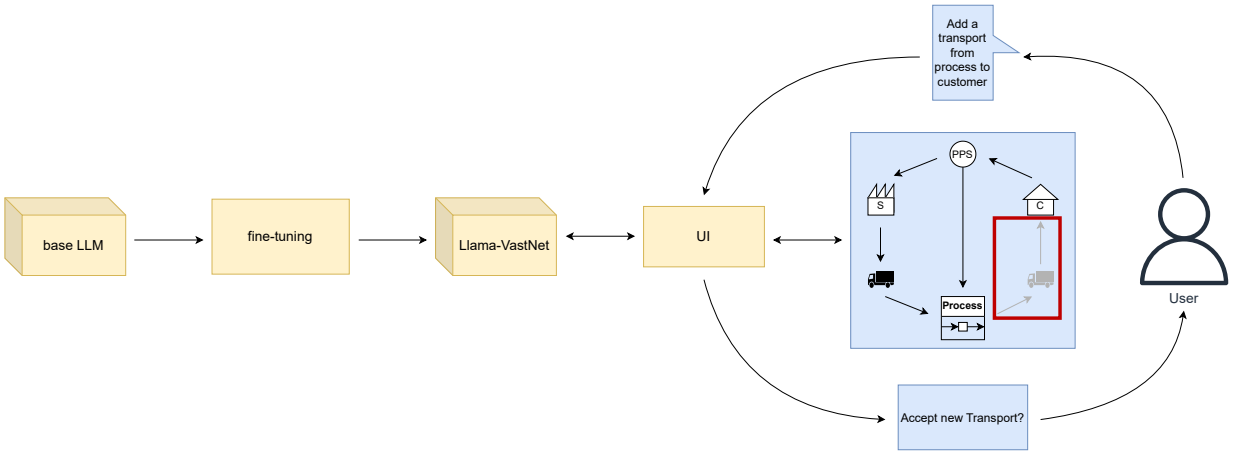

Figure 1: Architecture Overview. The user writes a request to the fine-tuned model. The VS is parsed and added to his request as context for the model. The model recognizes that it has to create a new node, transport and connects it to the nodes process and customer. It generates a function call to add the missing node and links to the VS. The user is asked to accept the changes proposed by the model.

2 RELATED WORK

This section first discusses fundamentals on LLMs (Section 2.1), followed by VSM - specific applications of machine learning on models in general (Section 2.3) and LLMs specifically (Section 2.4).

2.1 Large Language Model (LLM) Fundamentals

Recently, LLMs such as ChatGPT (OpenAI et al. 2023), LLaMA (Grattafiori et al. 2024) or DeepSeek (DeepSeekAI et al. 2025) have undergone a rapid development and have become powerful general problem solvers, capable of understanding and generating text. LLMs have been found to excel in various applications, including natural language processing, content summarization and generation, and even reasoning.

Formally, LLMs are neural networks with billions of parameters, which are trained with stochastic gradient descent to minimize the network's prediction error. The training process is typically split into two main phases: (1) Self-supervised pre-training, in which the model learns general language patterns by predicting missing parts of vast text corpora. (2) Instruction fine-tuning, where the model learns to follow human-like instructions. In this training phase, the model is repeatedly presented small tasks consisting of inputs, questions (instructions) and expected response. The model receives the combined input + question, and is tasked with the generation of the corresponding response. A loss function measures how much the model's output deviates from the target answer, and this loss is minimized through back-propagation. This process adjusts the models parameters so it can address user requests more accurately.

2.2 Generating Formal Languages with LLMs

LLMs support not only the generation of natural language but also of formal languages. The basis for this is sufficient exposure during pre-training, which is why, for example, the quality of generated code/software depends on the popularity of the programming language used (Sathvik et al. 2024), or why widely-used formats for process modeling such as BPMN (Köpke and Safan 2025) can be processed by LLMs to some extent.

For other, lower-resource domains with less training exposure, LLMs need to be adapted. This can be done in two fundamental ways:

1. Additional *knowledge* can be added to the prompt in form of instructions or few-shot examples. For example, nl2spec (Cosler et al. 2023) uses few-shot examples to condition the model to generate logical formulas from sentence fragments. Another, popular, approach called *retrieval-augmented generation (RAG) (Zhao et al. 2024)* utilizes a textual – often task-specific – knowledge base. Given a user question, relevant passages are detected and added to the LLMs prompt.
2. The *reasoning* ability, i.e. the capability to interpret domain-specific inputs and generate domain-specific outputs can also be improved, either by adding few-shot examples to the prompt which showcase the desired behavior (also known as *in-context learning* Brown et al. 2020), or by explicitly conducting an additional *instruction fine-tuning* on domain-specific input/output pairs (Yang et al. 2024). To do so, the research community has collected labeled datasets for various languages such as first-order logic (Yang et al. 2024) or SQL (Chen et al. 2024).

In this paper, we focus on the model's reasoning abilities (2.). Specifically, since VSs tend to be too large to add multiple examples to the prompt, we abstain from in-context learning and focus on *instruction fine-tuning* instead.

2.3 Machine Learning Assisted VSM

The integration of machine learning techniques in VSM has been explored in various ways, though few approaches leverage the instruction-following capabilities of LLMs to modify and generate VSs dynamically.

Our work builds upon prior studies that apply machine learning to VSM while extending the capabilities of existing models by enabling natural language interaction and a broader range of use cases.

Wrzalik et al. (2023) developed a model based on random forests (Breiman 2001) capable of suggesting corrections for broken VSs. While effective for very specific repair operations, their model was constrained in its applicability and lacked the ability to process a wide variety of user instructions. Furthermore, it did not support natural language interaction, which limits its accessibility to non-expert users.

Becker and Funke (2020) focused on predicting changes in Material Flow Networks using network embedding methods, such as Node2Vec. This was done to gain valuable insights for manufacturing and logistics optimization.

Vernickel et al. (2020) employed an XGBoost model (Chen and Guestrin 2016) to predict parameters for material flow simulations. While these studies demonstrate the utility of machine learning in industrial applications, they do not address direct user interaction or instruction-based VS modification.

2.4 LLM Usage with VSM

Geisthardt, Engel, and Gómez (2024) explored a game-based approach using a multi-modal LLM for VSM and lean management education. Their work highlights the potential of LLMs in enhancing understanding and engagement but does not focus on dynamic VS generation and modification.

Keskin et al. (2025) utilized an LLM to improve production information interfaces for process optimization. Their approach aligns with our objective of leveraging LLMs for enhanced user interaction, though their focus is on information retrieval rather than active modification of VSs.

Magnus and Venschott (2024) investigated the feasibility of using ChatGPT as a Lean Manufacturing consultant with mixed results. While their findings indicate the potential use of LLMs in lean manufacturing, it also showed that their approach with the current state of LLMs produces unreliable and unstable results. In our work, we use an open-weight LLM and limit its use to that of a VS modeling assistant in the hope of producing more stable and reliable results. To achieve this, we do not only prompt the LLM, but also use fine-tuning to enhance the LLMs capability to modify and correct of VSs directly based on natural language instructions.

3 APPROACH

To effectively leverage an LLM as a VSM assistant, it is essential to adapt the model to the specific terminology, structures, and reasoning patterns used in VSM and lean management as a whole. In our work, we investigate the feasibility of using fine-tuning to achieve this. Here we outline the methodology used. In Section 3.1, we describe the underlying language model selected as the foundation for fine-tuning, including the rationale for its selection and relevant architectural characteristics. Section 3.2 presents the composition of the training dataset, including the sources of VSM data, the annotation strategies used, and the preprocessing steps applied to ensure quality and consistency. Finally, Section 3.3 details the fine-tuning process, including hyper-parameter settings, evaluation criteria, and mechanisms employed to prevent over-fitting and enhance domain-specific performance.

3.1 Base Model

An LLM's input prompt is a sequence $\mathbf{x}_{1:n} = x_1, ..., x_n$, of words fragments (or *tokens*). Given $\mathbf{x}_{1:n}$, the model predicts possible output token sequences $\mathbf{y}_{1:m} = y_1, ..., y_m$ by computing the probability

$$P(\mathbf{y}_{1:m} \,|\, \mathbf{x}_{1:n}) = \prod_{i=1}^{m} P(y_i \,|\, \mathbf{x}_{1:n}, \mathbf{y}_{<i})$$

The LLM itself is basically a parametric function f_θ that iteratively predicts the next token, i.e. $P(y_i \,|\, \mathbf{x}_{1:n}, \mathbf{y}_{<i}) \approx f_\theta(\mathbf{x}_{1:n}, \mathbf{y}_{<i})$, with parameters θ. In most LLMs, f_θ follows the well-known architecture of so-called *trans-*

former decoders (Radford, Alec et al. 2018). Specifically, we use Meta's Llama-3.3-70B (Meta-Github 2024) model, which has 70 billion parameters. We chose this model as a foundation for this work because:

- The model is probably the most used open-weights model at the time and performs well on various benchmarks,
- Its parameters are publicly available for fine-tuning,
- The model supports a large input size of 128K tokens, which is required since the input VSs form input sequences of multiple thousands of tokens, and
- The model understands German, which is necessary since our work targets a German-speaking user base.

Training a model fundamentally works by collecting input-output pairs of input prompts $\mathbf{x}_{1:n}$ and corresponding outputs $\mathbf{y}_{1:m}$. In our case, $\mathbf{x}_{1:n}$ is a question regarding a VSs, and $\mathbf{y}_{1:m}$ is the desired answer, or a desired patch to the VSs. The model is iteratively presented an input, the well-known *negative log-likelihood (NLL)* loss function is computed:

$$\mathscr{L}(\theta) = -\sum_{i=1}^{m} log\ f_\theta(\mathbf{x}_{1:n}, \mathbf{y}_{<i})$$

The lower this loss, the more likely the LLM is to predict the desired output $\mathbf{y}_{1:m}$. $\mathscr{L}$ is minimized using stochastic gradient descent.

3.2 Dataset and Preprocessing

In order to fine-tune a LLM a sizable dataset of VSs is needed. To our knowledge, there is no such dataset publicly available. To solve this problem, we designed a generator which generates variations of, previously collected set of 70 VSs. Using this approach we created a dataset of 511 VSs, capturing a variety of VS modeling options. 460 VSs of the total dataset were used for fine-tuning the model. 51 were set aside for later evaluation.

In addition, we collected a dataset of 26 VSs. Each VS was paired with real user instructions. These instructions detailed changes that had to be applied to the VS, which we also included in the dataset as labels.

The VSs were stored in large JSON files that contained a lot of VSM unrelated information. To reduce the input size, only the most relevant information was extracted and put into a format that was similar to JSON, but with fewer parentheses and colons to further save input space. The VSs used for fine-tuning had 32 nodes on average.

3.3 The Llama-VastNet Training Procedure

Figure 2 illustrates our fine-tuning procedure: In each iteration, a random VS is selected from our training set (left), as well as a task from our pool of nine tasks (see Section 3.3.2). The VS is corrupted with a task-specific corruption function: For example, when selecting the task *Node Recovery*, a random node is removed from the VS, and the model is tasked with recovering the removed node. The result is a training sample containing:

- The corrupted VS.
- A *prompt*, which contains (1) a general text that introduces the model to the domain, to the specific data formats, and to its role as a VSM assistant, and (2) a task-specific instruction, which represents the user requests. Please consult the Appendix for an example prompt.
- The desired fix (represented as a function call and its parameters), which would repair the VS correctly.

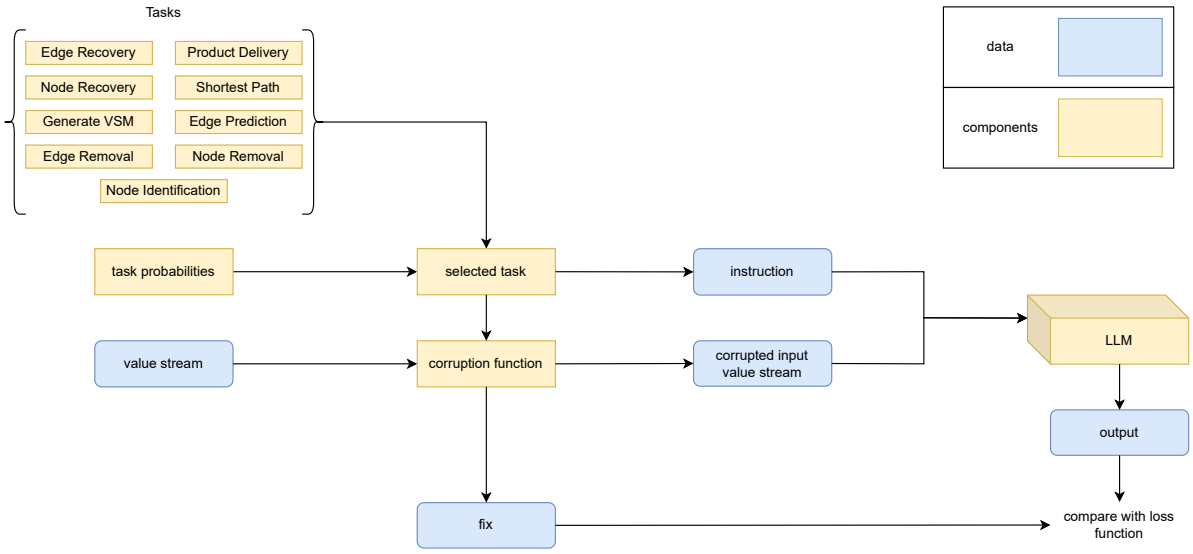

Figure 2: Overview of the VS usage while fine-tuning.

The LLM is passed the corrupted VS and the prompt, and the LLM's suggested fix is compared with the desired fix via a loss function, resulting in a correction to the model's weights. This training loop is applied repeatedly, priming the model to solve the VSM - specific training tasks.

3.3.1 Augmentation

Augmentation is a well-known technique in machine learning to improve the generalization capabilities of models in the ML domain (Shmuel et al. 2025), specifically when training set size is limited and over-fitting is an issue. Its idea is to randomly modify the input samples without altering their semantics, for example by applying noise to an input image.

We apply augmentation to our VSs: In each training step, we apply the following augmentations to our VSs before corrupting and feeding them to the model:

- The nodes and edges are shuffled (which does not change semantics, since nodes are referred to by IDs).
- The positions of nodes (which are displayed when drawing the VS in a tool's UI) are slightly perturbed by adding noise to the position (this was only applied in tasks where the model did not have to predict node positions).

These steps add more diversity to the training data which the LLM is exposed to in training.

3.3.2 Training-Tasks

To prime Llama-VaStNet to "understand" VSs, "good" training tasks should satisfy the following criteria: They should be (1) *challenging* for the model, i.e. they should enforce the model to understand the input data's syntax and semantics, but at the same time manageable, (2) *useful*, i.e. tasks should equip the model with capabilities required to solve real-world users' information needs, and (3) *automat able*, i.e. the corruptions and the desired fixes can be scripted.

Based on these criteria, we have defined nine VSM specific pre-training tasks (behind each name is the probability with which the respective task was chosen during training). The first category of tasks are concerned with recovering simple information about the VS without modifying it. They are targeted at equipping the model with reasoning capabilities essential to conducting more complicated repairs:

- **Node Identification (1%)**: Nodes in the VS are referred to by unique IDs, whereas users commonly refer to nodes by their type or name (e.g., "remove the storage"). Thus, we added this essential task, where a random node is selected, and the model is tasked with naming the node's ID, given the node's name.
- **Edge Prediction (3%)**: Another essential piece of information is whether two nodes are connected by a direct edge or not. Just like node identification, this task appears rather trivial (since the input VSs contains the list of edges), but we still keep it as a foundation task.
- **Product Delivery (8%)**: Here, the model was asked to identify all nodes that delivered input products to a randomly selected node.
- **Shortest Path (20%)**: Here, two interconnected nodes are selected, and the model is asked to output the shortest path between two nodes. All nodes on the path had to be generated.

Note that – since the above tasks address an information need about the VS rather than a repair – *no corruption* is applied to the VS in these cases. Other tasks, however, mimic errors by the user, and challenge the model to fix these errors. Thereby, we focus on simple corruptions that can be automated:

- **Edge Recovery (6%)**: A random edge is removed from the VS. The model is then asked to identify the missing edge.
- **Edge Removal (9%)**: A random edge was added to the VS, and the model is asked to identify and remove the extra edge.
- **Node Removal (20%)**: A random node and some random connecting edges are added to the VS. The model is then asked to remove the node and its edges.
- **Node Recovery (23%)**: A node and all its edges are removed. Afterwards the model is tasked with generating the missing node and its edges.

Finally, to teach Llama-VaStNet the "VS language" altogether, we mimic the self-supervised language modeling task:

- **Generate VS (10%)**: The model is tasked with generating the entire parsed VS.

Thus, we added this essential task to identify nodes by their types. Note that in this task, the VS is not corrupted.

3.4 Configuration

QLoRA was used to save computing power during fine-tuning. All layers were frozen. Rank 8 LoRA (Hu et al. 2022) adapters were applied to all query and key parameters. All the other parameters except the head model and the embedding layer were quantized (Hubara et al. 2018) to 4 bits. Gradient checkpointing was used to save GPU memory. The parameters for generation were set to the defaults. The fine-tuning was done over 15 epochs and took 4 days to complete on 5 A100 GPUs.

4 EVALUATION

In this chapter, we evaluate the performance of Llama-VaStNet with respect to its ability to manipulate VSs in response to user requests. The evaluation aims to assess not only the effectiveness of fine-tuning in improving task-specific performance, but also the model's ability to generalize to practical, real-world inputs. To this end, we conducted two experiments, each targeting a different aspect of model performance. To save computation, the tested models were again quantized to 4 bits, as for the fine-tuning.

In section 4.1, we compare Llama-VaStNet and the base model on the tasks used during fine-tuning. This experiment allows us to assess whether the model has successfully learned to perform the specific manipulations it was trained on, and to what extent fine-tuning has improved task accuracy relative to the base model.

In section 4.2, we shift the focus to real-world applicability by evaluating both models on a collection of actual user requests. This evaluation serves two purposes: First, it establishes a baseline for how well the base model performs on realistic input; and Second, it helps determine whether Llama-VaStNet shows measurable improvements in understanding and executing these user-driven modifications.

Finally, in section 4.3, we give some qualitative observations about the conducted experiments and try to interpret the results.

4.1 Fine-Tuning Task Performance

To see if our fine-tuning worked and that the model was capable of learning the tasks we had designed, we evaluated both Llama 3.3 and Llama-Vastnet on the fine-tuning tasks themselves. The VSs used in this evaluation had not previously been used for fine-tuning or testing. This was done to get an accurate reading of the models performance on unknown data.

For both models, all of the 51 VSs collected in this dataset, were used once for each fine-tuning task. The model could generate tokens up to twice the target length from the fine-tuning task. If it generated an end-of-sequence token, the generation stopped early. The generated outputs of the models were compared to the targets. This was done at the character level using an edit distance. The character edit distance is calculated by counting each character that must be inserted, deleted, or substituted to make the two strings the same. This number was then divided by the longer of the two strings to normalize the results. Table 1 shows the results of this experiment.

Table 1: Evaluation on fine-tuning tasks. Each score represents the normalized character edit distance of the models output to the label. Bold indicates the better score.

task	LLama 3.3	Llama-VaStNet
Node Identification	0.334	**0**
Edge Prediction	0.243	**0.065**
Product Delivery	0.768	**0.029**
Shortest Path	0.779	**0.34**
Edge Recovery	0.484	**0.045**
Edge Removal	0.523	**0.054**
Node Removal	0.678	**0.063**
Node Recovery	0.565	**0.226**
Generate VSM	0.833	**0.608**
average	0.578	**0.159**

4.2 User Requests

To evaluate on the VSs with user requests, the models were prompted in the same way as for the evaluation on the fine-tuning tasks. The only differing from the latter in the user instructions. The model output was validated with the predefined label of the sample. An output was considered correct if all function calls and their parameters were identical to those of the label, with a few exceptions. The position of a node was considered correct as long as it was close to the desired position (about 300 pixels or less). The id of a newly created node was ignored as long as it was not already occupied by an existing node and the connection edges correctly pointed to it. The order of the function calls was ignored as long as they ended up in the same end result. Any additional generated text, that was not part of a function call was ignored. This mostly helped the base model, which tended to explain its function usage with text and sometimes executed the same function call again after the explanation.

A user request included multiple changes. Each required creation or removal function call for a node or a edge was counted as one change. A change request was translated into a removal and a creation

function call changed with the new attribute. The dataset of 26 samples contained a total of 125 changes. The samples were divided into the three categories: create, change and delete. Table 2 shows the result of the evaluation.

Table 2: Evaluation of on real user requests of Llama 3.3 and Llama-VaStNet. The test data is divided into the three categories: create, change and remove. Each score in these categories is the percentage of correct changes made by the model. The total column is the total percentage of correct changes across all categories. The average is calculated across the three subcategories. Bold indicates the better score.

model	create	change	remove	total	average
LLama 3.3	26.7%	19.2%	40.6%	32.8%	28.8%
Llama-VaStNet	**36.7%**	**76.9%**	**49.3%**	**52%**	**54.3%**

4.3 Qualitative Observation

The results of the two experiments provide strong evidence that the fine-tuning procedure was effective in improving the LLMs ability to understand and manipulate VSs according to user instructions. In both experiments, the Llama-VaStNet consistently outperformed the base model, although with some notable limitations that point to future improvement opportunities.

In the first experiment, Llama-VaStNet achieved a significantly better normalized edit distance compared to the base model. This demonstrates that it not only learned the logic of the manipulations but also internalized the expected output format. The base model, by contrast, frequently violated formatting requirements, often producing explanatory text despite being prompted not to. This severely impacted its scores but simultaneously highlights a key success of the fine-tuning: it effectively aligned the model with both task structure and output constraints.

The second experiment further validates the effectiveness of the fine-tuning in enhancing the models manipulation capabilities. Llama-VaStNet outperformed Llama 3.3 in all three categories, with a particularly strong margin in the change category. This suggests that the model benefits from having access to attribute information within the input context. This likely compensated for the lack of explicit exposure to some attributes during fine-tuning, allowing the model to infer the necessary changes based on the data it received. However, the current function setup required models to perform a change by deleting the entire element and recreating it with the new attribute an unnecessarily complex and error-prone process. Introducing a dedicated function for changing attributes directly could significantly simplify this operation and likely improve performance for both models.

In contrast, the performance in the create category was comparatively lower. One probable cause is that many creation tasks involved attributes and values that were not present in the fine-tuning data. In such cases, even when the model generated an appropriate structure, the absence of correct or complete attribute information led to the output being marked incorrect. This highlights a limitation in generalization to unseen attribute-value pairs.

The remove category also posed difficulties. Here, both models struggled, though the fine-tuned model showed a modest improvement. The removal function available to the models only deleted nodes, not their associated edges. This design choice was intentional to encourage the model to understand and model the connections between nodes and edges but it also introduced a layer of complexity that likely degraded performance. The models had to explicitly remove all associated edges in addition to the node itself. It is likely that with removal function that deletes both a node and its associated edges, both models would have achieved better results in this category.

Overall, while the fine-tuning clearly improved the base models ability to adhere to output format, and correctly manipulate VSs across diverse tasks, the absolute performance remains insufficient for practical deployment. The fine-tuned model, though directionally promising, is not yet reliable or accurate enough to act as a standalone assistant for VSM tasks.

5 CONCLUSION

In this work, we presented a fine-tuned LLM Llama-VaStNet tailored to VSM tasks, with the aim of building an intelligent assistant capable of interpreting and manipulating VSs based on user input. Through a series of targeted training tasks and evaluations, we demonstrated that our approach leads to clear improvements over the base model on training tasks and in responding to real-world user requests involving VS modifications. While these results are promising, they also highlight the current limitations of the approach. Despite the observed gains, the model's performance is not yet sufficient to serve as a fully reliable VSM assistant in practical settings. The model still struggles with extensive and complex requests as well as ambiguity in user input, often requiring it to make assumptions that can lead to suboptimal or incorrect modifications.

To advance this line of research, several avenues for future work emerge. First, refining the design of training tasks to better reflect the complexity and variability of real VSM scenarios may lead to further performance improvements. Secondly, extending the training data to include simulation-based data from the VSs could enhance the model's ability to make sensible change suggestions. Additionally, rethinking the model's usage paradigm from it needing to produce whole VSs changes to a more interactive agent-based approach, were VSM unrelated details are left to the software. Embedding the model within an environment that supports structured access to relevant data (e.g., through tool APIs or dynamic prompts) may significantly improve performance and reliability.

Ultimately, this work establishes a foundation for a LLM-based VSM assistant, and outlines a clear path toward more robust and intelligent process improvement tools.

ACKNOWLEDGMENTS

This work was funded by the German Federal Ministry of Research, Technology and Space (Project "VaStNet", ID 01IS24015C).

A APPENDIX

This is the general part of the prompt (also known as the *system prompt*).

```
You are a value stream mapping expert. You recieve requests from
users and reply in a usfull way. A value stream map is graph with
nodes and links between them. A link connects two nodes by
referencing their ids, with the properties "from" and "to". Here
you are given the input value stream map which the user bases his
requests on:
```

The following is an example for a task-specific instruction, for the task *Node Recovery*:

```
The value stream map is missing a node and its associated links.
First, create the missing node. Then, create the links that
connect the generated node to others in the value stream map.
Create only the one missing node and its associated links.
```

Since our research targets a German-speaking customer base, the original (German) prompts were translated to English for presentation in this paper.

REFERENCES

Becker, T., and T. Funke. 2020. "Machine Learning Methods for Prediction of Changes in Material Flow Networks". *Procedia CIRP* 93:485–490.

Breiman, L. 2001. "Random Forests". *Machine Learning* 45(1):5–32.

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, *et al*. 2020. "Language Models are Few-Shot Learners". In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*. December 6th-12th, virtual, 1877-1901.

Chen, P. B., F. Wenz, Y. Zhang, D. Yang, J. Choi, N. Tatbul, *et al*. 2024. "BEAVER: An Enterprise Benchmark for Text-to-SQL". *arXiv preprint* https://doi.org/10.48550/ARXIV.2409.02038.

Chen, T., and C. Guestrin. 2016. "XGBoost: A Scalable Tree Boosting System". In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. August 13th-17th, San Francisco, USA, 785-794.

Cosler, M., C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel. 2023. *nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models*, 383–396. Cham: Springer Nature Switzerland.

DeepSeekAI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, *et al*. 2025. "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning". *arXiv preprint* https://doi.org/10.48550/ARXIV.2501.12948.

Geisthardt, M., L. Engel, and J. M. Gómez. 2024. "Simulation Game Based Teaching of Value Stream Analysis and Design with Multimodal Large Language Models (MLLMs) Assistance". In *Proceedings of the SAP Academic Community Conference 2024 (D-A-CH)*. September 3th-4th, Vienna, Austria, 9-16.

Grattafiori, A., A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, *et al*. 2024. "The Llama 3 Herd of Models". *CoRR* https://doi.org/10.48550/ARXIV.2407.21783.

Hou, X., Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, *et al*. 2024. "Large Language Models for Software Engineering: A Systematic Literature Review". *ACM Transactions on Software Engineering and Methodology* 33(8):1–79.

Hu, E. J., Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, *et al*. 2022. "LoRA: Low-Rank Adaptation of Large Language Models". In *The Tenth International Conference on Learning Representations, ICLR*. April 25th-29th, Virtual Event.

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. 2018. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". *journal of machine learning research* 18(187):1–30.

Jackson, I., M. Jesus Saenz, and D. Ivanov. 2023. "From Natural Language to Simulations: Applying AI to Automate Simulation Modelling of Logistics Systems". *International Journal of Production Research* 62(4):1434–1457.

Keskin, Z., D. Joosten, N. Klasen, M. Huber, C. Liu, B. Drescher *et al*. 2025. "LLM-Enhanced Human–Machine Interaction for Adaptive Decision-Making in Dynamic Manufacturing Process Environments". *IEEE Access* 13:44650–44661.

Köpke, J., and A. Safan. 2025. *Efficient LLM-Based Conversational Process Modeling*, 259–270. Springer Nature Switzerland.

Magnus, C. S., and M. Venschott. 2024. "Towards a GPT-Based Lean Manufacturing Consultant for Manufacturing Optimization". *Procedia CIRP* 130:167–176.

Meta-Github 2024. "Meta-Llama 3.3 Github". https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md. Accessed: Aug. 13, 2025.

OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, *et al*. 2023. "GPT-4 Technical Report". *CoRR* https://doi.org/10.48550/ARXIV.2303.08774.

Radford, Alec et al. 2018. "Improving Language Understanding by Generative Pre-Training". https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. Accessed: Aug. 14, 2025.

Rother, M., and J. Shook. 2003. *Learning to See: Value Stream Mapping to Add Value and Eliminate Muda*. Boston: Lean enterprise institute.

Sathvik, J., J. W. Wu Jie, and H. Fard Fatemeh. 2024. "A Survey on LLM-based Code Generation for Low-Resource and Domain-Specific Programming Languages". *arXiv preprint* https://doi.org/10.48550/ARXIV.2410.03981.

Shmuel, A., O. Glickman, and T. Lazebnik. 2025. "Data Augmentation for Deep Learning Regression Tasks by Machine Learning Models". *CoRR* https://doi.org/10.48550/ARXIV.2501.03654.

Vernickel, K., L. Brunner, G. Hoellthaler, G. Sansivieri, C. Härdtlein, L. Trauner, *et al*. 2020. "Machine-Learning-Based Approach for Parameterizing Material Flow Simulation Models". *Procedia CIRP* 93:407–412.

Wrzalik, M., J. Eversheim, J. Villmow, A. Ulges, D. Krechel, S. Spieckermann *et al*. 2023. *Value Stream Repair Using Graph Structure Learning*, 15–32. Cham: Springer Nature Switzerland.

Yang, Y., S. Xiong, A. Payani, E. Shareghi, and F. Fekri. 2024. "Harnessing the Power of Large Language Models for Natural Language to First-Order Logic Translation". In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. August 12th-14th, Bangkok, Thailand, 6942-6959.

Zhao, P., H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, *et al*. 2024. "Retrieval-Augmented Generation for AI-Generated Content: A Survey". *CoRR* https://doi.org/10.48550/ARXIV.2402.19473.

AUTHOR BIOGRAPHIES

MICHA SELAK is a research assistant in the department of Design Computer Science Media at RheinMain University of Applied Science in Wiesbaden, Germany. He is a former IT-Consultant and is part of the Learning and Visual Systems working group. His research interests include Automated Code Analysis, Code Representation Learning and LLM-Assisted VSM. His e-mail address is micha.selak@hs-rm.de.

DIRK KRECHEL is a professor with RheinMain University of Applied Sciences in Wiesbaden since 2005. He graduated from TU Kaiserslautern (2003 PhD in computer science, 1997 diploma in computer science) and has been head of application development for knowledge management solutions of the SER Solutions Deutschland GmbH (2001-2005, Bonn). Dirk's research fields are machine learning for text, semantic technologies and medical applications. His publication record includes >60 peer-reviewed papers. His e-mail address is dirk.krechel@hs-rm.de.

ADRIAN ULGES has been a professor with RheinMain University of Applied Sciences in Wiesbaden since 2013. He graduated from TU Kaiserslautern (2009 PhD in computer science, 2005 diploma in computer science) and has worked with Google as an intern (2005, Mountain View) and as a visiting scientist (2011, Zurich). Adrian's research focuses on machine learning for text, and has been awarded with a Google Research Award in 2010. His publication record includes >50 peer-reviewed papers. His e-mail address is adrian.ulges@hs-rm.de.

SVEN SPIECKERMANN, Ph.D., is Chief Executive Officer at SimPlan, Hanau, Germany, mainly working the research and development activities of SimPlan. Since 1992, he has been participating in over 300 simulation projects and various joint research initiatives. Additionally, he has been giving lectures in simulation at the Technical University of Braunschweig since 1995, at the Technical University of Darmstadt since 2008, and at the KIT in Karlsruhe since 2012. He has published several papers on simulation, simulation-based optimization and related topics. His email contact is sven.spieckermann@simplan.de.

NIKLAS STOEHR is a software developer at SimPlan AG, Hanau, Germany, where he has been contributing for five years. He holds an M.Sc. in Business Informatics from the Technical University of Darmstadt. His interests focus on VSM and making simulation more accessible, aiming to enhance efficiency and usability in software-driven decision-making processes. His email contact is niklas.stoehr@simplan.de.

ANDREAS LOEHR is a software developer at SimPlan AG, Hanau, Germany. He holds a M.Sc. in Mathematics from Goethe University Frankfurt. His primary focus lies in leveraging cutting-edge machine learning research to improve various aspects of the VSM process. His email contact is andreas.loehr@simplan.de.