Leveraging Generative AI for Enhancing Domain-Driven Software Design

G-H. Wiegand; F. Stepniak; P. Baier

Link to the Paper:
https://pabair.github.io/publications

Introduction:

Domain-Driven Design (DDD) remains pivotal for building software that closely aligns with customer requirements by modeling domain concepts accurately. The intricate process of creating domain metamodels, however, traditionally relies on manual input from system designers, necessitating expertise and significant time investment. The rise of generative AI offers new opportunities to streamline these efforts by automating parts of the metamodel creation process. This paper investigates the use of generative AI, specifically a fine-tuned version of Code Llama, for the automated generation of JSON objects representative of domain-specific constructs.

Our research focused on the feasibility and effectiveness of training generative AI to generate syntactically correct JSON objects based on minimal prompts, a process crucial for supporting iterative software development in DDD. By fine-tuning the model on real-world DDD data and employing techniques such as 4-bit quantization and Low-Rank Adaptation (LoRA), we achieved robust results even on consumer-grade hardware. The findings underscore the potential of such models to serve as foundational tools in the DDD process, enhancing productivity and reducing resource expenditure.

The practical relevance of this work is embodied in the Domainlifecycles Code Generator (DCG). DCG functions as an AI-driven assistant integrated into the Analysis and Modeling phase (1) of the Domainlifecycle process, where it automates the creation of domain-specific language (DSL) components. As depicted in Figure 1, this phase is the first in a series of agile steps that include generating the code framework, implementing business logic, and refining the model through feedback. By streamlining the analysis and modeling phase, DCG enhances workflow efficiency, supporting the rapid and precise development of adaptable, business-aligned software solutions.

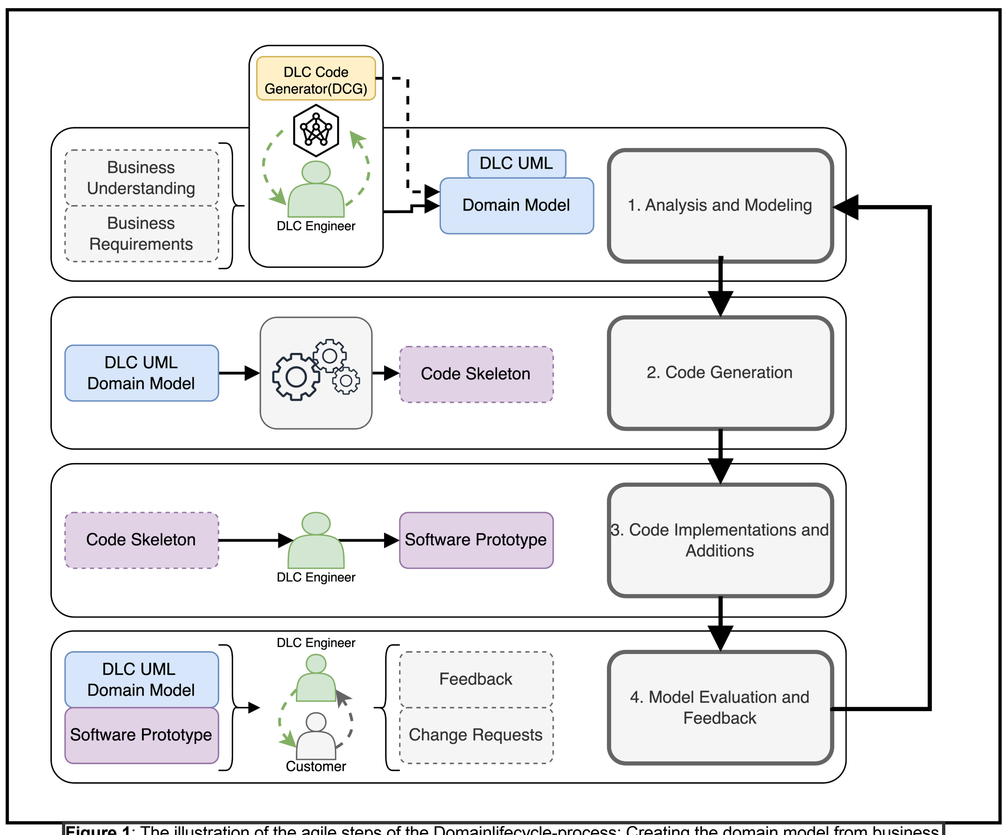

Figure 1: The illustration of the agile steps of the Domainlifecycle-process: Creating the domain model from business information (1. Analysis and Modeling), generating the code framework (2. Code Generation), formulating the logic in the code (3. Code Implementation) and receiving feedback and new business insights through presentation to the customer (4. Model Evaluation and Feedback).

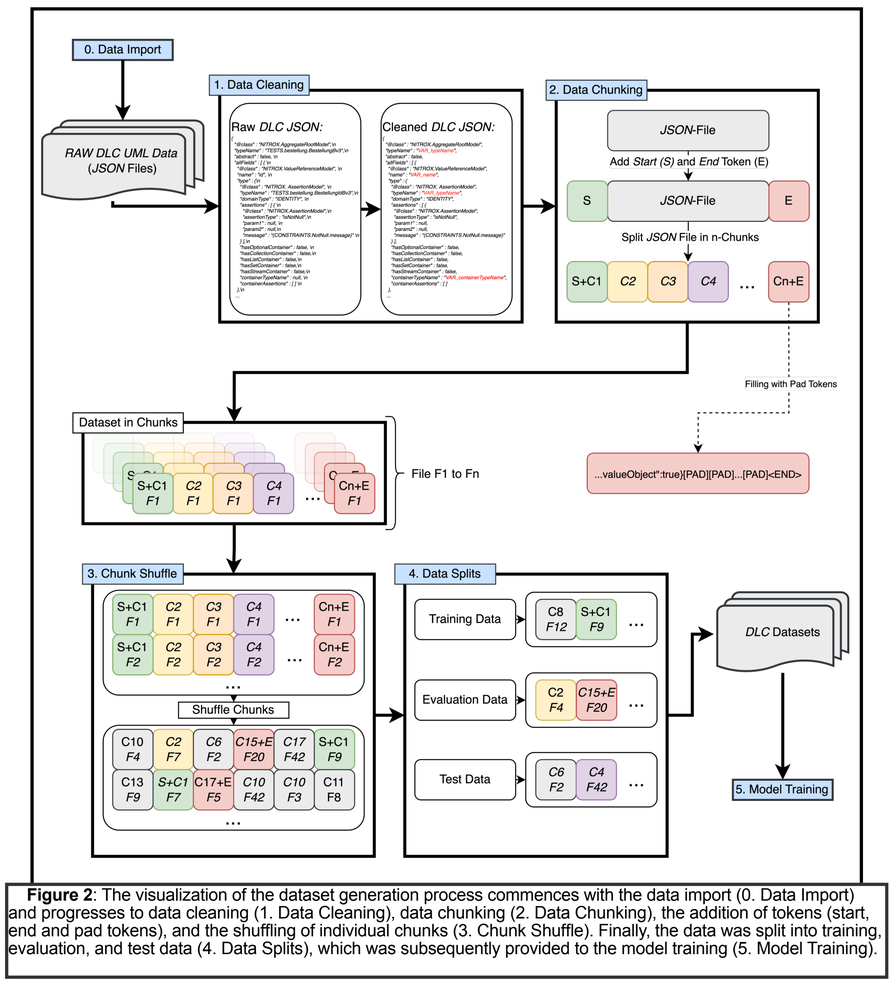

Figure 2: The visualization of the dataset generation process commences with the data import (0. Data Import) and progresses to data cleaning (1. Data Cleaning), data chunking (2. Data Chunking), the addition of tokens (start, end and pad tokens), and the shuffling of individual chunks (3. Chunk Shuffle). Finally, the data was split into training, evaluation, and test data (4. Data Splits), which was subsequently provided to the model training (5. Model Training).

Methods:

To develop an initial generative AI assistant for Domain-Driven Design (DDD), this project trained a next-token predictor to generate domain models in JSON format for specified classes, focusing on machine-readable JSON output as a primary requirement.

Data for this project came from a sample project and a client project, requiring strict confidentiality measures. Data was anonymized and abstracted during preprocessing, which also involved formatting and preparing it for model training (see Figure 2). The resulting data was used to fine-tune a CodeLlama-7B model using Low-Rank Adaptation (LoRA) and quantization, allowing the model to train efficiently on only 11GB of VRAM.

Model performance was assessed using the BLEU score and training loss, recorded via the Hugging Face Trainer. These metrics also guided hyperparameter tuning, where a weighted score combining BLEU and inverse loss (1 - loss) helped identify optimal settings. Final training was completed using these tuned parameters, resulting in the model used for further validation.

To ensure the model could generate syntactically valid and meaningful DDD outputs, a three-phase assessment process was applied. First, evaluation metrics were reviewed to confirm baseline performance. Next, 100 JSON objects were generated from prompts and checked for syntactic correctness with a JSON parser. Finally, a qualitative analysis was conducted on selected samples to identify any potential issues. This multi-phase evaluation provided a comprehensive understanding of the model's suitability for generating structured, machine-readable DDD outputs.

Results:

In the final training phase, the model achieved a loss of 0.0337 on the evaluation data and 0.0393 on the test data, alongside BLEU scores of 0.9924 on evaluation data and 0.9918 on test data. The minimal differences in loss and BLEU scores between evaluation and test datasets suggest that the model training was effective and consistent across both sets.

To assess the model's performance, JSON samples generated by the model were reviewed in two distinct categories based on the clarity of the input prompts. In cases labeled "Experimental," the prompts did not clearly specify the target object for the JSON output, whereas "Clear" samples were based on explicit prompts that defined the object to be generated. Out of 100 samples generated, 81 were successfully parsed by a JSON parser with only minor post-processing, indicating they were syntactically correct. Notably, all 50 samples within the "Clear" category were syntactically accurate, suggesting that the model produces optimal results when prompts are specific.

A final analysis of errors and incorrectly generated samples provided further insights, identifying several patterns that may guide improvements in future model versions.

Overall, the findings demonstrate that the model can produce machine-readable JSON objects when given appropriate prompts. Fine-tuning on a GPU with sufficient memory yielded strong results, and dataset abstraction reduced complexity, allowing for the use of customer data. This initial model version shows potential for integration into applications, with attention to the identified biases.

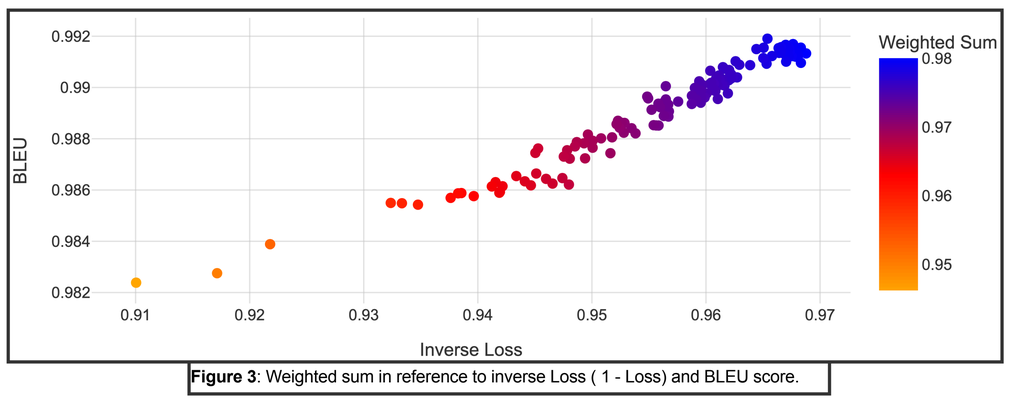

Figure 3: Weighted sum in reference to inverse Loss (1 - Loss) and BLEU score.

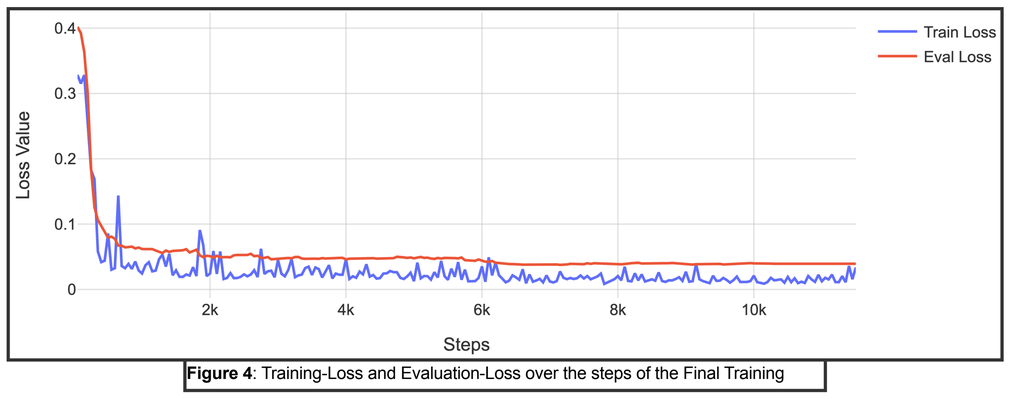

Figure 4: Training-Loss and Evaluation-Loss over the steps of the Final Training

esentri

Hochschule Karlsruhe
University of Applied Sciences