Machine Learning Surrogate Models for Performance Prediction with Architectural Models

Sebastian Weber*, Vincenzo Pace†, Thomas Weber†, Jörg Henß*, Robert Heinrich‡

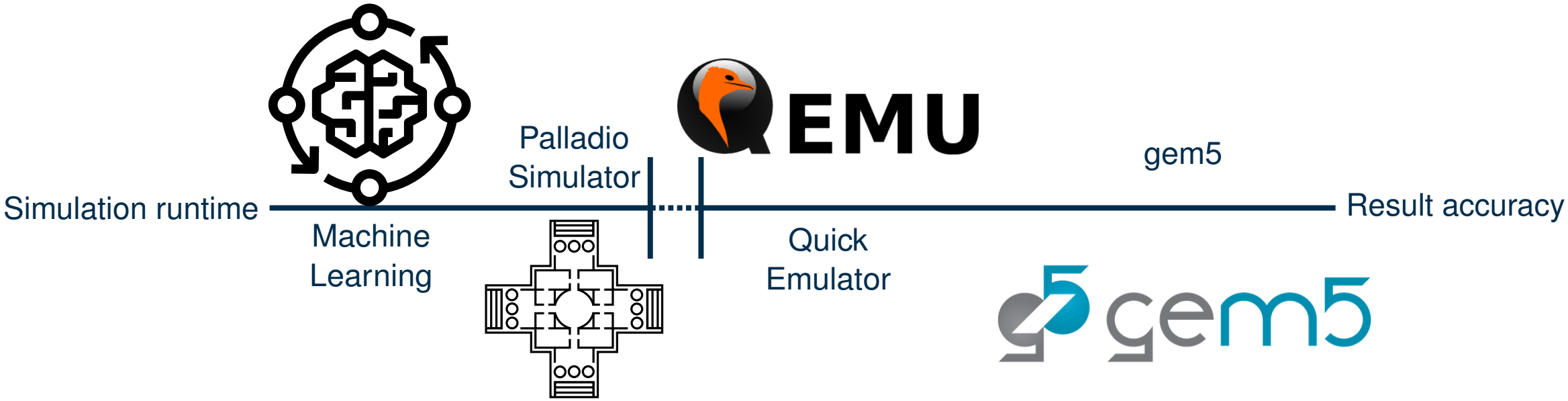
Thomas Weber† | 4th November 2025

*FZI Research Center for Information Technology
†Karlsruhe Institute of Technology
‡Ulm University

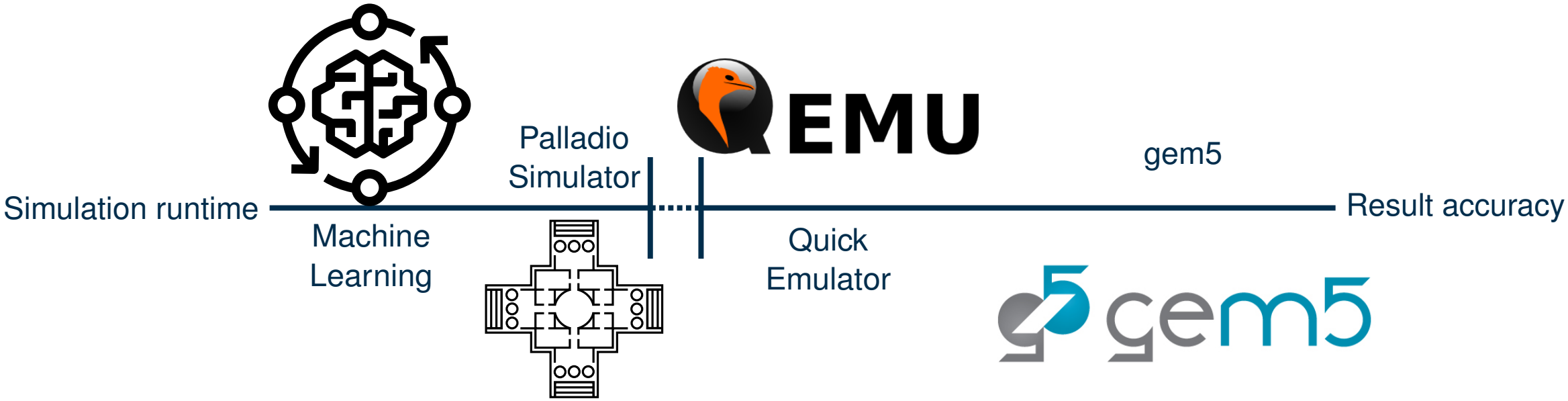
FZI KIT

SDQ Software Design and Quality

Motivation


Simulation runtime
Machine Learning
Palladio Simulator
QEMU
Quick Emulator
gem5
Result accuracy
gem5

Motivation



Tradeoff between Simulation Runtime and Accuracy

- Start with machine learning as a surrogate for Palladio
- Future work will evaluate larger models as alternative for more detailed simulators

Background

Palladio

- Architecture performance modeling framework
- Supports coarse grained modeling already at design time
- Predicts metrics like response time and CPU usage
- Treated as a black box to generate the training data

Background

Palladio

- Architecture performance modeling framework
- Supports coarse grained modeling already at design time
- Predicts metrics like response time and CPU usage
- Treated as a black box to generate the training data

Palladio Component Model (PCM)

- Repository, Assembly, Resource Environment, Deployment, Usage
- Instances based on PCM-Metamodel
- Text-based PCM specifies the elements according to a DSL similar to the PCM-Metamodel

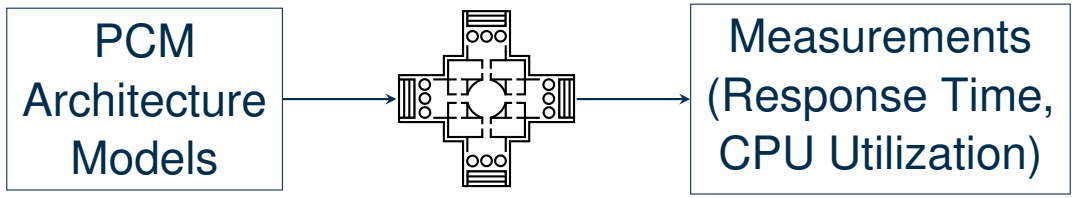
Background

Palladio

- Architecture performance modeling framework
- Supports coarse grained modeling already at design time
- Predicts metrics like response time and CPU usage
- Treated as a black box to generate the training data

Palladio Component Model (PCM)

- Repository, Assembly, Resource Environment, Deployment, Usage
- Instances based on PCM-Metamodel
- Text-based PCM specifies the elements according to a DSL similar to the PCM-Metamodel



Text-based Palladio Component Model (TPCM)

```
import std::definitions

repository minimalRepo {
  datatype PrimitiveInt INT

  interface IMinimalService {
    op doSomething(param PrimitiveInt)
  }

  component MinimalComponent {
    provides pMinimal IMinimalService
    requires cpu ICPU

    seff pMinimal.doSomething {
      cpu.process(<param.VALUE>)
    }
  }
}
```

```
system minimalSystem {
  provides sMinimal minimalRepo::IMinimalService ->
  assembly A1 minimalRepo::MinimalComponent
}

resourceenvironment minimalEnv {
  container C1 {
    processing CPU CPUResource
  }
}

allocation minimalAllocation {
  minimalSystem::A1 -> minimalEnv::C1
}

usage minimalUsage {
  "Minimal Scenario" population(<1>) thinkTime(<0.1>) {
    minimalSystem::sMinimal.doSomething (<42>)
  }
}
```

Machine Learning

Embeddings

- Convert text into numerical vectors that capture meaning
- Term Frequency-Inverse Document Frequency (TF-IDF)
  - Algorithm to calculate embeddings
  - Counts word frequency in and across documents and gives them according vectors
- Bidirectional encoder representations from transformers (BERT)
  - Large Language Model for the generation of embeddings
  - Learns context-sensitive, deep semantic representations

Machine Learning

Embeddings

- Convert text into numerical vectors that capture meaning
- Term Frequency-Inverse Document Frequency (TF-IDF)
  - Algorithm to calculate embeddings
  - Counts word frequency in and across documents and gives them according vectors
- Bidirectional encoder representations from transformers (BERT)
  - Large Language Model for the generation of embeddings
  - Learns context-sensitive, deep semantic representations

Machine Learning (ML)

- Model Types
  - Random Forest
  - Support Vector Machine
  - Artificial Neural Network
  - Linear Models
- Hyperparameters are parameters configuring the training process and ML model properties, but not the weights

Related Work

Highly Configurable Systems

- Deep (Ha and Zhang 2019) and adversarial (Shu et al. 2020) learning to improve accuracy with limited data
- Prediction of execution time and scalability

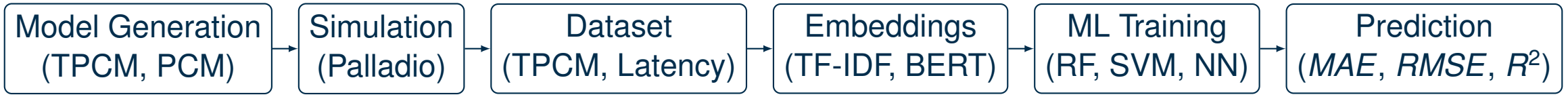
High Performance Computing

- Benchmark comparing eleven machine learning methods for modeling the performance of four representative scientific applications on four HPC platforms (Malakar et al. 2018)
- Comparing shallow and deep neural networks for three algorithms with different computations and memory-access patterns (Mankodi, Bhatt, and Chaudhury 2020)

Code-level predictors

- DeepTLE (Zhou et al. 2019) uses deep learning on tokenized code sequences to predict performance without executing code

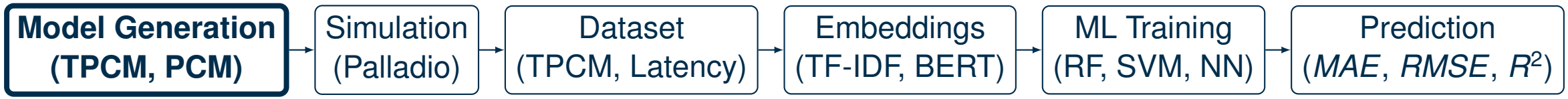
Approach — Overview



Surrogate Model Training Pipeline

- Generate architecture models in the TPCM-format
- Transform them to the PCM-format and simulate them
- Aggregate the TPCM models and simulation results for the training dataset
- Calculate the embeddings for the dataset
- Train the different ML model

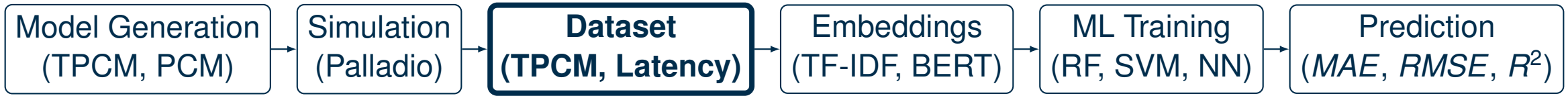
Approach — Model Generation



Model Generator

- Generate random number of model elements in given bounds
- Randomly connecting the model elements to suitable other model elements, e.g., a component to an interface
- Model elements were generated first from the repository, then assembly, then deployment, then usage
- No further domain knowledge was used, e.g., generating the system model based on a random DAG
- TPCM-format was chosen as input format for the ML models due to its better human readable structure
- To be able to simulate the models they had to be converted to the PCM-format

Approach — Dataset

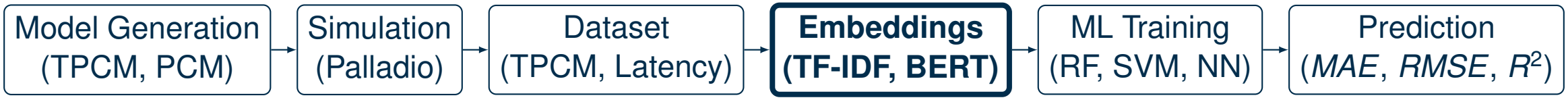


Dataset

- Successfully simulated (T)PCM models with their average response time
- Skewed distribution of response times
- Some models were syntactically correct but could not be simulated, therefore they were excluded

Metric	Value
Number of samples	15,529
Min input length (characters)	3,231
Max input length (characters)	20,457
Average input length (characters)	8,915.40
Min avg_resp_time (s)	0.000000
Max avg_resp_time (s)	157.782044
Average avg_resp_time (s)	0.479739

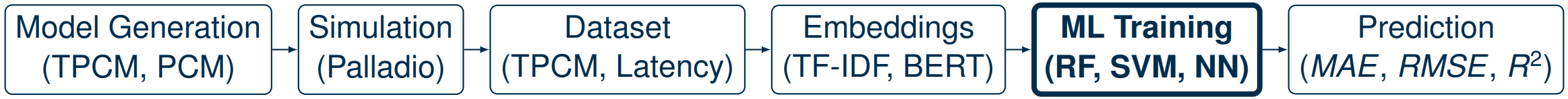
Approach — Embeddings



Embedding Calculation

- Calculated once
- TF-IDF as simple baseline, BERT for contextual embeddings
- 512 token limit of BERT solved by chunking input model with sliding window and averaging resulting vectors

Approach — ML Training



Model Types

- Random Forest
- Support Vector Machine
- Artificial Neural Network
- Linear Models (Ridge and Lasso Regression)

Model Training

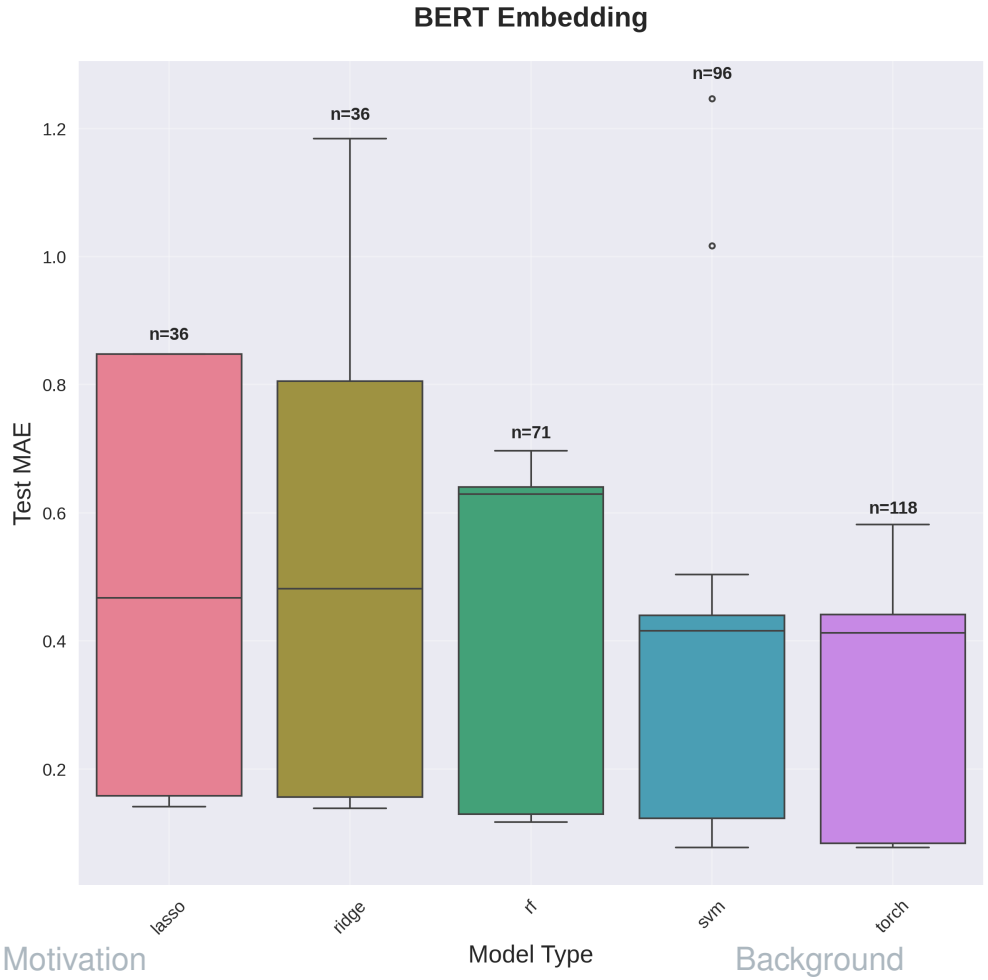
- Hyperparameters value ranges tuned manually
- All hyperparameters fixed except one
- Repeat training runs per configuration 3 times and average results
- Compare each model to dummy predicting average

Evaluation

Goals

- Prediction quality compared to the Palladio results
- Prediction quality of the different model types
- Prediction quality of the two different embeddings

Evaluation



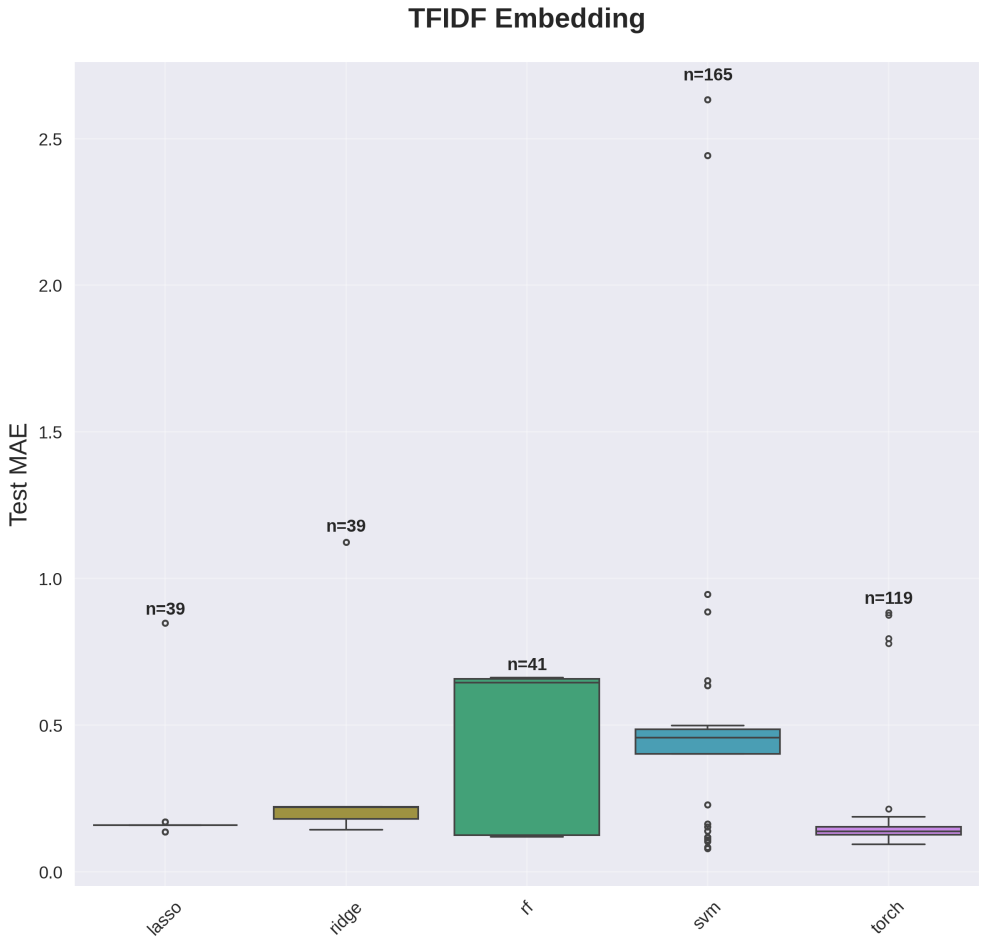
Results BERT

- Models perform similarly
- SVM and ANN (torch) slightly ahead
- Mean absolute Error (MAE) worse than expected

Evaluation

Results TF-IDF

- Strong linear performance
- Large svm outliers
- Slightly better than BERT (different scalings of the MAE due to the outliers)



Evaluation

Goals

- Prediction quality compared to the Palladio results
- Prediction quality of the different model types
- Prediction quality of the two different embeddings

Results overall

- High Mean absolute Error given the small TPCM models
- No clear best model type
- TF-IDF embeddings performed slightly better than BERT embeddings

Evaluation

Threats to validity

- **Data Generation Bias:** Randomly generated TPCM models do not reflect realistic architectural designs, limiting external validity and representativeness
- **Simulation Failures and Filtering:** Roughly 40% of generated models failed simulation or were excluded due to invalid structure, potentially biasing the dataset toward simpler or semantically trivial configurations
- **Limited Variability in Target Metrics:** Many valid simulations yielded near-zero response times, restricting the range of the dependent variable and weakening learning signals
- **Computational Constraints:** Lack of GPU resources and restricted hyperparameter searches may have limited the optimization and tuning of ML models
- **Metric Sensitivity:** Differences between MAE and R^2 interpretations could distort conclusions, as R^2 penalizes outliers more heavily in a noisy, low-signal dataset

Evaluation

Limitations

- **Low Information Density in Models:** The generated models were randomly connected, which resulted in potentially huge parts of the models being irrelevant for the performance prediction
- **Insufficient Data Volume and Quality:** The dataset size and randomness restricted generalization and amplified the noise–signal imbalance in model training
- **Low Interpretability of Complex ML Models:** One benefit of Palladio is the easy interpretation of the simulation results, which the complex ML models do not have
- **Overfitting and Instability:** Ensemble and neural models showed high variance across runs, often fitting noise rather than meaningful architectural patterns
- **Limited Transferability:** Models trained on synthetic TPCM data probably do not generalize to real-world architectures or larger-scale systems

Conclusion

Results

- End-to-end pipeline for ML-based performance prediction from TPCM models
- Automatic generator for random TPCM models
- Explored TF-IDF and BERT embeddings with various ML models
- Predictive accuracy remained low

Conclusion

Results

- End-to-end pipeline for ML-based performance prediction from TPCM models
- Automatic generator for random TPCM models
- Explored TF-IDF and BERT embeddings with various ML models
- Predictive accuracy remained low

Future Work

- Improve the random generation of TPCM models
- Use encoders capable of embedding TPCM models without chunking
- Try graph-based representations of the TPCM models
- Integrate explainable AI approaches to ensure the correct conclusion can be drawn
- Try few-shot or zero-shot prompting with LLMs

Conclusion

Results

- End-to-end pipeline for ML-based performance prediction from TPCM models
- Automatic generator for random TPCM models
- Explored TF-IDF and BERT embeddings with various ML models
- Predictive accuracy remained low

Future Work

- Improve the random generation of TPCM models
- Use encoders capable of embedding TPCM models without chunking
- Try graph-based representations of the TPCM models
- Integrate explainable AI approaches to ensure the correct conclusion can be drawn
- Try few-shot or zero-shot prompting with LLMs

Contact

- Sebastian Weber
- sebastian.weber@fzi.de

Literatur I

[1] Huong Ha and Hongyu Zhang. “DeepPerf: Performance Prediction for Configurable Software with Deep Sparse Neural Network”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. May 2019, pp. 1095–1106. DOI: 10.1109/ICSE.2019.00113. URL: https://ieeexplore.ieee.org/document/8811988/?arnumber=8811988 (visited on 11/10/2024).

[2] Preeti Malakar et al. “Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications”. In: *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. Nov. 2018, pp. 33–44. DOI: 10.1109/PMBS.2018.8641686. URL: https://ieeexplore.ieee.org/document/8641686/?arnumber=8641686 (visited on 11/19/2024).

[3] Amit Mankodi, Amit Bhatt, and Bhaskar Chaudhury. “Evaluation of Neural Network Models for Performance Prediction of Scientific Applications”. In: *2020 IEEE REGION 10 CONFERENCE (TENCON)*. Nov. 2020, pp. 426–431. DOI: 10.1109/TENCON50793.2020.9293788. URL: https://ieeexplore.ieee.org/document/9293788/?arnumber=9293788 (visited on 09/20/2024).

References

Literatur II

[4] Yangyang Shu et al. “Perf-AL: Performance Prediction for Configurable Software through Adversarial Learning”. In: *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Bari Italy: ACM, Oct. 5, 2020, pp. 1–11. ISBN: 978-1-4503-7580-1. DOI: 10.1145/3382494.3410677. URL: https://dl.acm.org/doi/10.1145/3382494.3410677 (visited on 11/10/2024).

[5] Meiling Zhou et al. “Deeptle: Learning code-level features to predict code performance before it runs”. In: *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2019, pp. 252–259.

References

Acknowledgements

The HAL4SDV project is co-funded by the Chips Joint Undertaking (Chips JU) and National Authorities under grant agreement n° 101139789.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or National Authorities. Neither the European Union nor the granting authorities can be held responsible for them.

Co-funded by
the European Union

References