Exploring the Benefits of Just-in-time Model Replacement

Nils Strassenburg
nils.strassenburg@hpi.de
Hasso Plattner Institute
Potsdam, Germany

Boris Glavic
bglavic@uic.edu
University of Illinois Chicago
Chicago, USA

Tilmann Rabl
tilmann.rabl@hpi.de
Hasso Plattner Institute
Potsdam, Germany

ABSTRACT

With the rise of large language models (LLMs), organizations have access to general-purpose models through high-level APIs. Many organizations now outsource simple, repetitive tasks such as sentiment classification, support ticket labeling, or churn risk identification to LLMs to save on data collection and preparation, model selection, training, and deployment. However, such tasks can often be solved equally well by much simpler models that are several orders of magnitude more cost- and resource-efficient. In recent work, we propose *just-in-time model replacement (JITR)*. JITR fully automates the process of switching between an LLM and a cheaper, task-specific surrogate model, thus retaining the ease-of-use and low development cost of outsourcing tasks to an LLM while significantly reducing the inference cost of repetitive tasks. JITR is enabled by advanced model search algorithms that search a model repository to find and fine-tune an appropriate model for a task. In this demonstration, we present *JITR-Explore*, a tool for exploring the trade-offs involved in JITR. Through an interactive dashboard, users can explore cost savings from using JITRs compared to LLMs across a variety of tasks, controlling parameters such as the models used and the number of expected requests.

PVLDB Reference Format:
Nils Strassenburg, Boris Glavic, and Tilmann Rabl. Exploring the Benefits of Just-in-time Model Replacement. PVLDB, 19(12): 4674 - 4677, 2026.
doi:10.14778/3827998.3828094

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hpides/poodle/tree/demo.

1 INTRODUCTION

LLMs combine state-of-the-art machine learning (ML) performance with ease-of-use via standardized APIs. From a user's perspective, leveraging an LLM requires no ML expertise, comes without upfront model development costs, and does not require data collection, labeling, or preprocessing. This makes LLMs attractive for offloading simple, recurring data-processing tasks. As an example, imagine a medium-sized non-tech company that has to classify customer reviews or forward support tickets to the right employee. While this required the development of a custom model in the past, nowadays companies increasingly offload these tasks to LLMs via API calls or their data-warehouse provider [5, 9].

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828094

The drawback of offloading narrow, long-standing tasks to LLMs is their high cost. A possible solution is to develop a specialized surrogate model for the target use case, but this comes at the expense of usability and requires ML expertise and dataset curation.

To bridge this gap between easy-to-use but expensive LLMs and cheap but difficult-to-use custom models, we proposed *just-in-time model replacement (JITR)* [11]. Analogous to just-in-time compilation, where slow interpreted code is replaced with equivalent compiled code at runtime, JITR automatically detects when an LLM handles simple repetitive tasks, develops a surrogate model, and transparently replaces the LLM with the surrogate. This preserves the ease-of-use of an LLM but at the low cost of a small model.

Analyzing the time spent in each JITR stage, we determined that the most critical factor for effective, scalable JITR is cheap and fast surrogate model development. This allows JITR to amortize monitoring and model development overheads and replace LLMs as early as possible. The key enabler for this is an efficient model store that supports fast model search. Such a model store must efficiently store, index, retrieve, and search through large collections of models and their metadata, which is a data management challenge that is actively discussed in the community but far from solved [3, 7].

In this paper, we introduce *JITR-Explore*, an interactive dashboard for exploring the effects of JITR. Starting from a predefined scenario, attendees can modify parameters such as the LLM in use, the target use case, prompt length, or expected request volume, and observe the resulting impact on cost savings, throughput, and accuracy. The remainder of this paper is organized as follows. Section 2 provides an overview of JITR [11], followed by an introduction to *JITR-Explore* in Section 3. Section 4 describes the proposed demonstration use cases, and Section 5 concludes the paper.

2 JUST-IN-TIME MODEL REPLACEMENT

Consider a scenario in which a company stores customer reviews in a database table and assesses customer satisfaction by issuing a semantic query that infers the sentiment of every review. A baseline for implementing such a query involves selecting an LLM, writing a prompt for the task, embedding each row into the prompt, sending it to the LLM, waiting for a response, and extracting the result.

With JITR, we follow the four-stage approach shown in Figure 1, which we implemented in our JITR prototype *Poodle* [11].

Data Collection & Task Detection (Stage 1). In the first stage, we monitor the baseline workflow described above and feed the results to *Poodle*'s *Data Collector*. We collect data such as task type and input data modality, which we use to detect long-standing, recurring tasks, and the LLM request-response pairs, which we use to compose a labeled training dataset for a task.

Model Development (Stage 2). Once the *Data Collector* identifies a recurring task, it forwards all related data to *Poodle*'s *Model Generator*, which develops a specialized surrogate model and deploys

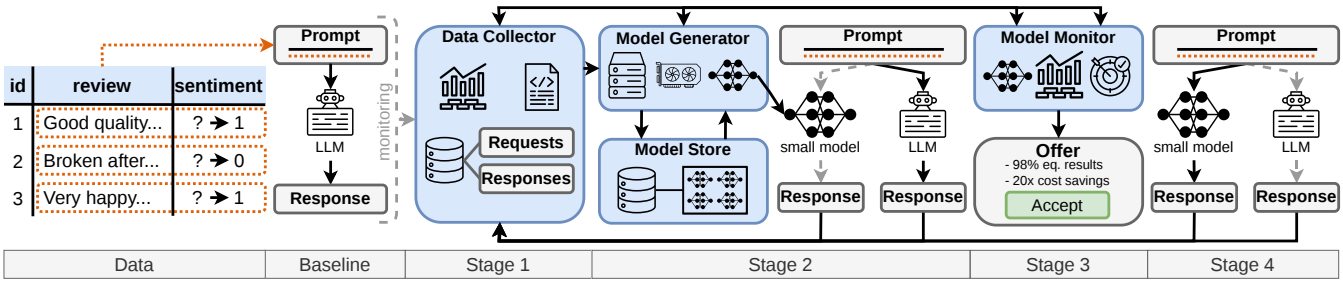

Figure 1: Four stage JITR workflow, including *Poodle*'s components (blue background).

it as an alternative to the LLM. The *Model Store* supports the model development process by providing access to previously developed models, and stores new models developed during JITR.

Replacement Decision (Stage 3). The *Model Monitor* monitors the surrogate model's behavior by routing a fraction of requests to both the LLM and the surrogate model to estimate cost savings, latency improvements, and differences in accuracy. If the surrogate model performs well, the *Model Monitor* offers the user to switch to the surrogate model for future requests.

Surrogate Deployment (Stage 4). If the user accepts, the *Model Monitor*'s offer, *Poodle* routes all new incoming requests to the surrogate model. Additionally, the *Model Monitor* continues to monitor the surrogate model's performance by sending a fraction of the requests to the LLM and comparing the LLM's and the surrogate model's outputs to detect deteriorating performance or data drift.

In our previous paper [11], we show that: (1) JITR amortizes the custom model development overhead and leads to significant cost savings; (2) JITR reduces the inference time and latency even if the initially used LLM is small; (3) and that for sufficiently narrow tasks, small models developed with JITR achieve competitive accuracy. We refer the reader to our *Poodle* paper [11] for details, but briefly review task detection and model development.

Task Detection. *Poodle* detects tasks using wrapper prompts. Assuming the user request consists of a prompt and input data, a wrapper prompt extends the user prompt and instructs the LLM to produce two outputs. The first output is the response to the user prompt, combined with the input data; the second output is the metadata needed for JITR. In *Poodle*'s standard implementation, every LLM request is extended by a wrapper prompt. For this demo, a fraction parameter controls the share of wrapped requests.

Model Development. *Poodle*'s *Model Generator* develops models by combining model search [8] and model fine-tuning. *Poodle*'s *Model Generator* uses the labeled dataset collected by the *Data Collector* to issue a model search query to the *Model Store*. Given the dataset, the *Model Store* uses *Alsatian*'s [10] baseline model search algorithm to identify the most promising candidate for fine-tuning. In a second step, the *Model Generator* uses the LLM-labeled dataset to fine-tune the model retrieved from the *Model Store*. The rationale for this two-stage process is that using model search to identify suitable base models for fine-tuning outperforms standard fine-tuning in terms of accuracy, required data, and development time [8, 11], which allows us to replace the LLM with a high-quality surrogate model early-on.

3 JITR-EXPLORE

In this section, we first describe *JITR-Explore*'s user interface in Section 3.1, provide details on our cost estimation in Section 3.2, and discuss estimating other metrics and current limitations of *JITR-Explore* in Section 3.3. The *Poodle* prototype and experimental code were primarily developed by the authors with AI assistance. The UI was generated using AI tools, and its outputs were validated against ground-truth measurements.

3.1 User Interface

Figure 2 shows *JITR-Explore*'s user interface. At the top, *JITR-Explore* shows the selected scenario, plots, and key statistics for comparing the baseline LLM approaches in *Poodle*. The plots show a cost comparison and a break-even analysis. The statistics shown by the system include expected cost savings, the break-even point, the earliest time to switch models, expected accuracy, expected throughput, and the time required to process the expected number of requests. Next to the metrics, symbols indicate whether the displayed values are from external sources (e.g., model prices as provided by the LLM provider), measured by us (e.g., accuracy and throughput), or calculated (e.g., cost savings).

Attendees begin with a predefined or empty scenario and modify it to update the plots and statistics. The area in the green solid box provides an overview of different models with their input (p_{in}) and output (p_{out}) token prices. Attendees can add custom models, select the LLM baseline models, and the type of surrogate model.

In the area highlighted by the orange dashed box, the *Use Case* panel lets attendees specify input data (x_{input}), a prompt (x_{prompt}), and output (x_{output}) data. The *Task Detection* panel lets them specify whether a wrapper prompt is used, and if so, the content ($x_{wrapper}$), the expected output ($x_{wrapper\_out}$), and the fraction of wrapped requests (α). In the area highlighted by the red dotted box, the *Requests* panel lets attendees specify the expected number of requests (R) and the threshold after which we switch to the surrogate model (R_{detect}). The *Validation* panel lets them configure the fraction of requests sent to each model during monitoring (β).

In the *Model Development Panel*, attendees set model development costs, choose the model development technique, and explore model development internals by pressing the *Details* button. The model details view shows multiple panels, each simulating a specific model search technique. Figure 3 shows the panel for Alsatian's baseline model search, which estimates each model's fine-tuning

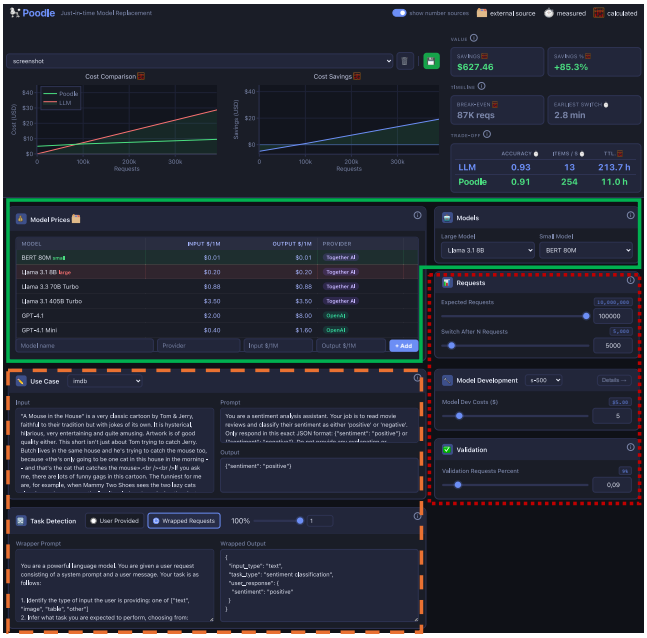

Figure 2: *JITR-Explore*'s user interface.

performance via a proxy score [10]. The start button runs a simulation based on actual measurement traces, with the timer in the top-right corner indicating the model search time. First, the system collects 1000 LLM-labeled items. It then sequentially generates proxy scores for each candidate model and sorts the models by performance. In the example shown, the top model reaches an estimated accuracy of 0.9 on the labeled data, the second model 0.72, the third is being evaluated, and the remaining models are queued.

3.2 Cost Estimation

LLM Approach Cost Estimation. We define the cost of processing an average request as the sum of input and output token costs. A request consists of a system prompt x_{prompt} and user input x_{input}; the model produces an output x_{output}. Let $t(x)$ denote the token count of a text x estimated by the *tiktoken* library [6], and let p_{in} and p_{out} denote the per-token prices for input and output tokens. The cost of a single request with model M is:

$$c_M = [t(x_{\text{prompt}}) + t(x_{\text{input}})] \cdot p_{\text{in}} + t(x_{\text{output}}) \cdot p_{\text{out}}$$

For a workload of R requests, the total cost scales linearly:

$$C_M(R) = R \cdot c_M$$

The cost of the baseline with LLM M is thus $C_{\text{Base}} = C_{\text{M}}(R)$.

***Poodle* Cost Estimation.** The total cost of *Poodle*, C_{Poodle}, comprises four components: (i) task detection during the initial request phase, (ii) developing the small model, (iii) serving requests with the fine-tuned small model, and (iv) ongoing quality monitoring.

$$C_{\text{Poodle}} = C_{\text{detect}} + C_{\text{dev}} + C_{\text{serve}} + C_{\text{monitor}}$$

Task Detection (C_{detect}). Before replacing the LLM M with a surrogate model, *Poodle* processes R_{detect} requests using M. A fraction

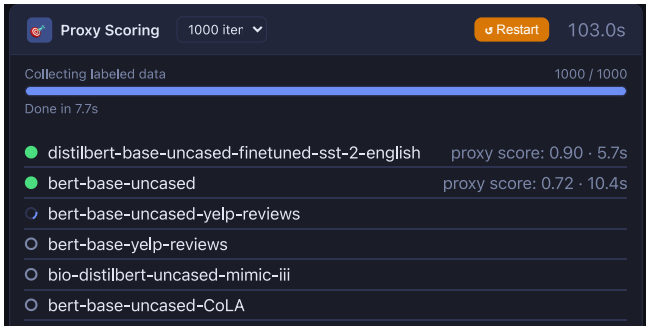

Figure 3: Excerpt of Model development details.

α of these are *wrapped* requests, the remaining $1 - \alpha$ are not:

$$C_{\text{detect}} = C_{\text{M}}^{\text{wrap}}[\alpha \cdot R_{\text{detect}}] + C_{\text{M}}[(1-\alpha) \cdot R_{\text{detect}}]$$

Processing a wrapped request with model M includes the cost of the wrapper prompt x_{wrapper} (input) and additional output $x_{\text{wrapped\_output}}$:

$$c_M^{\text{wrap}} = [t(x_{\text{prompt}}) + t(x_{\text{wrapper}}) + t(x_{\text{input}})] \cdot p_{\text{in}} + t(x_{\text{wrapped\_out}}) \cdot p_{\text{out}}$$

Model Development (C_{dev}). We estimate the development cost of the surrogate model SM by multiplying the GPU-hours required by the on-demand price of an AWS EC2 G5 instance (with an NVIDIA A10G GPU), which closely matches our setup, at \$1.20/hr [1].

$$C_{\text{dev}} = t_{\text{dev}} \cdot \$1.20/\text{hr}$$

Model Serving (C_{serve}). Once the small model is deployed, it handles the remaining $R - R_{\text{detect}}$ requests:

$$C_{\text{serve}} = C_{\text{SM}}[R - R_{\text{detect}}]$$

Quality monitoring (C_{monitor}). To detect quality degradation, *Poodle* periodically forwards a fraction β of requests to the LLM:

$$C_{\text{monitor}} = C_{\text{LLM}}[\beta \cdot (R - R_{\text{detect}})]$$

3.3 Measurements and Limitations

Statistics. *JITR-Explore* dynamically updates costs based on user inputs as described above. Measuring model accuracy, throughput, and model development time for the given configuration takes too long for interactive updates. We instead provide estimates for a subset of configurations based on our actual measurements using a server with an NVIDIA RTX A5000 GPU.

Combination of Measured and Reported Numbers. For some metrics, we combine measured results to extrapolate new metrics, in other cases, we combine measured results with reported numbers by other sources. For example, when reporting BERT results, we measure accuracy, throughput, and model development time. However, it is difficult for us to estimate the true cost of hosting a BERT model as an LLM-provider. We rely on the reported prices of the model closest to our locally measured model. To report the minimum time to process all requests, we combine our model development time and throughput measurements.

Evaluation Scope. Our current prototype handles classification tasks, and our evaluation covers IMDB sentiment classification using BERT as a surrogate model. In future work, we plan to expand

both the evaluation datasets and task types, covering more complex semantic operators, such as summarization, entity resolution, and text2sql. BERT and similar surrogate architectures are unlikely to have been pre-trained on common benchmarking datasets such as IMDB, giving a realistic lower-bound accuracy estimate when training the surrogate model on LLM-labeled data [2]. LLMs, however, likely incorporate such datasets into their pre-training data, giving them an advantage on these benchmarks.

4 DEMONSTRATION

With JITR, we bridge the gap between easy-to-use but expensive LLMs and difficult-to-use but efficient custom models by automatically replacing LLMs with cheaper models. Using *JITR-Explore*, attendees explore cost savings, throughput improvements, and gain an intuition for the performance-accuracy trade-off when using JITR. Throughout the demonstration, attendees analyze the impact of different numbers of requests, model combinations, and model development methods. Other configuration settings, such as custom prompts, custom models, or different task-detection methods, are fully functional in *JITR-Explore* but are omitted for space reasons.

Initial Configuration. Attendees start the demo either by selecting one of the predefined scenarios or by starting the configuration from scratch. Let us assume we currently use GPT-4.1 with a price point of $2 for 1M input tokens and $8 for 1M output tokens. We further expect approximately 50,000 requests to the LLM and a use case similar to IMDB sentiment classification [4]. By default, we detect tasks by wrapping all incoming requests, sending 10% of the requests for validation, using the *base* model development approach, and switching after 5,000 requests to a BERT model.

Analyzing Results. Once we have fully defined our scenario, attendees analyze the costs of the LLM approach and our JITR prototype *Poodle* for an increasing number of requests in the *Cost Comparison* plot, and *Poodle*'s cost savings in the *Cost Savings* plot. Additionally, *JITR-Explore* shows key statistics on the right. Attendees will observe that: (1) the expected savings for this scenario are approximately $29, (2) the LLM-only approach and *Poodle* break even in terms of cost after 9,000 requests, (3) with the current model development approach, we can switch to a cheaper model after eleven minutes (the *time-to-switch*), (4) and that the surrogate model is competitive, achieving an accuracy of 0.89 with a throughput of 254 items/s. Starting from this scenario, attendees can now analyze how different factors influence the reported numbers.

Modifying Number of Requests. Increasing the number of requests to 1M and then to 10M leads to a recomputation of C_{Base} and C_{Poodle}, resulting in updated cost plots as well as adjusted savings of approximately $700 and $7,000, respectively.

Choosing a Different LLM. Changing the LLM to the smaller *LLama 8B* model triggers a re-estimation of C_{Base}, decreases the cost savings to $630, and shifts the break-even point to 12,000 requests. Additionally, measured accuracy and throughput numbers are available for *LLama 8B*, showing that *Poodle*'s surrogate model has competitive accuracy and 20× higher throughput.

Analyzing Effects of Model Search. We next reduce the number of requests to 500K and fix the surrogate model architecture. Changing *Poodle*'s model development approach affects all metrics except the baseline's accuracy and throughput. While cost differences are marginal, the effects on the time-to-switch, the surrogate model's accuracy, and the time to process all requests are significantly impacted. When adjusting the model development approach from *base* which takes a standard BERT model and fine-tunes it on the LLM generated labels to *S-Naive* which searches for the best model by fine-tuning all BERT models in the model store, attendees observe an increased surrogate accuracy of 0.92 but a steep increase for the time-to-switch (53 minutes) and the processing time for all requests (1.4 hours). Switching to *S-500*, which approximates the search [10, 11], accuracy only decreases by 1%, but the time-to-switch is now less than 3 minutes and also total processing time for all requests is significantly reduced.

5 CONCLUSION

We introduce *JITR-Explore*, an interactive user interface for exploring cost savings, throughput improvements, and accuracy trade-offs achieved by automatically replacing LLMs with custom surrogate models using just-in-time model replacement (JITR). Attendees can explore a variety of different scenarios to understand the impact of key parameters including the LLM used, the expected number of LLM requests, and the choice of model search technique. Our demonstration shows that JITR achieves significant cost savings at competitive accuracy when backed by an efficient model search system for transfer learning such as Alsatian [10].

ACKNOWLEDGMENTS

This work was funded by the German Research Foundation (ref. 414984028 and ref. 556566056) and NSF awards IIS-2420577 and IIS-2420691.

REFERENCES

[1] Amazon Web Services. 2026. Amazon EC2 On-Demand Instance Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. Accessed: March 2026.
[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.
[3] Peizhen Guo, Bo Hu, and Wenjun Hu. 2022. Sommelier: Curating DNN models for the masses. In *Proceedings of the 2022 International Conference on Management of Data*. 1876–1890.
[4] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 142–150.
[5] McKinsey & Company. 2024. Gen AI in Customer Care: Using Contact Analytics to Drive Revenues. https://www.mckinsey.com/capabilities/operations/our-insights/operations-blog/gen-ai-in-customer-care-using-contact-analytics-to-drive-revenues. Accessed: March 2026.
[6] OpenAI. 2026. tiktoken: A Fast BPE Tokeniser for OpenAI's Models. https://github.com/openai/tiktoken. Accessed: March 2026.
[7] Koyena Pal, David Bau, and Renée J. Miller. 2025. Model Lakes. In *Proceedings of the 28th International Conference on Extending Database Technology (EDBT)*. 985–995. https://www.openproceedings.org/2025/conf/edbt/paper-255.pdf
[8] Cedric Renggli, André Susano Pinto, Luka Rimanic, Joan Puigcerver, Carlos Riquelme, Ce Zhang, and Mario Lučić. 2022. Which model to transfer? finding the needle in the growing haystack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9205–9214.
[9] Snowflake. 2026. Snowflake Text Classification. https://docs.snowflake.com/en/sql-reference/functions/classify_text-snowflake-cortex. Accessed: March 2026.
[10] Nils Strassenburg, Boris Glavic, and Tilmann Rabl. 2025. Alsatian: Optimizing Model Search for Deep Transfer Learning. *Proc. ACM Manag. Data* 3, 3 (2025), 127:1–127:27. https://doi.org/10.1145/3725264
[11] Nils Strassenburg, Boris Glavic, and Tilmann Rabl. 2025. Poodle: Seamlessly Scaling Down Large Language Models with Just-in-Time Model Replacement. arXiv:2512.05525 [cs.DB] https://arxiv.org/abs/2512.05525