Using Large Language Models for Hyperparameter Optimization

Michael R. Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, Jimmy Ba
University of Toronto, Vector Institute

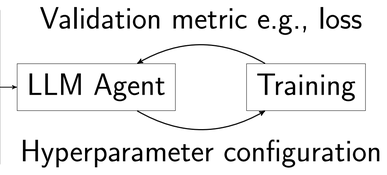
VECTOR INSTITUTE

UNIVERSITY OF TORONTO

You can just ask **large language models (LLMs)** which **hyperparameters** to use, and it works pretty well!

LLM-Assisted Hyperparameter Optimization

Initial prompt: You are helping tune hyperparameters to minimize loss... this is our search space... formulate your output as JSON.



LLMs for hyperparameter optimization. In this sequential decision making setup.

- We prompt an LLM with a description of the problem and the search space.
- The LLM then outputs a set of hyperparameters to evaluate.
- The environment executes a training run with the hyperparameters, returning a validation metric that is used to re-prompt the LLM.

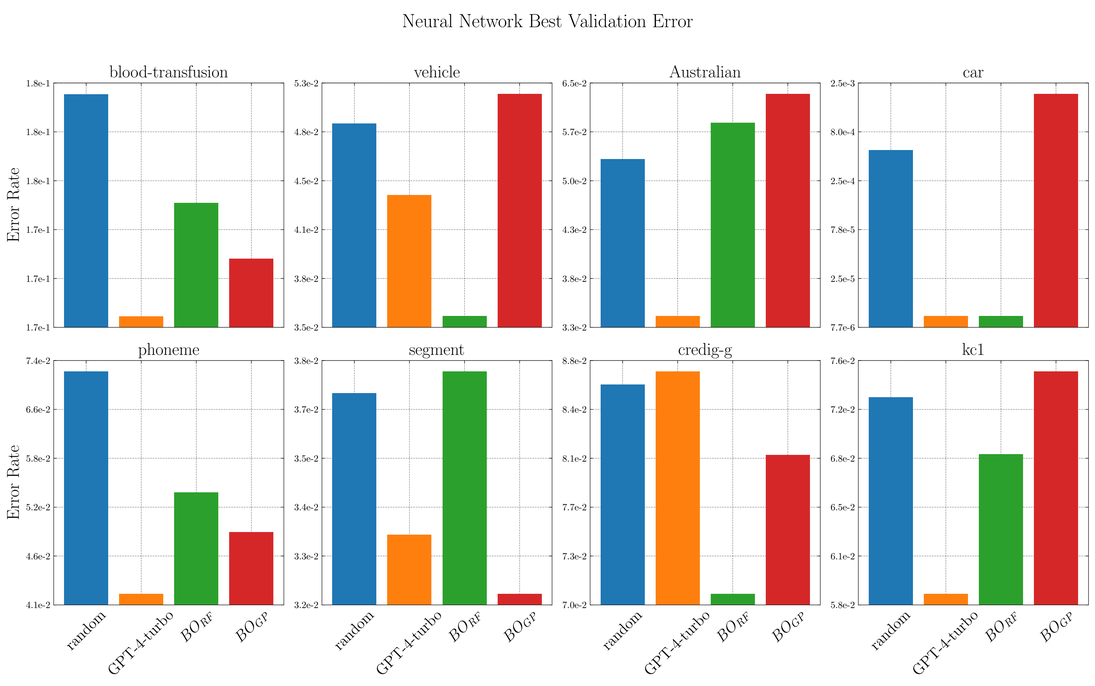
We consider two ways to structure the prompt:

1. Chat prompting [User: <Machine learning problem description>. Provide a config in JSON format. Config: (Assistant: <generated hyperparameter configuration> User: The validation loss was <L>. Provide the next config in the same format.) x N]
2. Single message prompting [User: <Machine learning problem description>. Here is what we have tried so far: <Config 1> <Loss 1> <Config 2> <Loss 2> ... <Config N> <Loss N> Provide the next config in JSON format. Config:]

Results

We summarize the performance of various HPO algorithms versus random search on 8 datasets and 4 search spaces tuning hyperparameters for logistic regression, SVMs, random forests, and neural networks. Bayesian optimization uses Gaussian process (*GP*) and random forest (*RF*) surrogate models. For all tasks, we use 10 function evaluations.

Model	Beats Random (↑)	Median Change (↑)	Mean Change (↑)	Mean Rank (↓)
GPT-4 Turbo	81.25%	13.70%	19.83%	2.42
GPT-4	68.75%	4.58%	8.54%	3.48
GPT-3.5 Turbo	43.75%	-0.82%	-13.58%	3.84
Bayes Opt$_{RF}$	56.25%	2.11%	5.86%	3.45
Bayes Opt$_{GP}$	50.00%	-0.01%	-8.28%	3.80



Minimum validation error tuning neural networks on HPOBench

Reasoning

We prompt the LLM to output its reasoning with the following: "Write two lines as follows: Analysis: Up to a few sentences describing what worked so far and what to choose next Config:"

Table: Chain-of-thought reasoning prompting may offer modest improvement versus prompting a model to output the next set of hyperparameters immediately.

Model	Beats Random (↑)	Median Change (↑)	Mean Change (↑)
GPT-4 Turbo CoT	81.25%	13.70%	19.83%
GPT-4 Turbo	81.25%	15.58%	21.23%
GPT-4 CoT	68.75%	4.58%	8.54%
GPT-4	65.62%	3.43%	11.55%
GPT-3.5 Turbo CoT	43.75%	-0.82%	-13.58%
GPT-3.5 Turbo	40.62%	-3.19%	-117.12%

Qualitatively, LLMs reasoning is a useful summary of experimental results and the relative importance of different hyperparameters.

GPT-4-Turbo: *The lowest loss so far is with Config 9, which suggests that a combination of a slightly deeper tree (max_depth: 15), a moderate number of features (max_features: 0.5), and a higher number of samples at the leaf nodes (min_samples_leaf: 4) and for splitting (min_samples_split: 8) is beneficial. The loss has not improved significantly with changes to the max_depth beyond the default, so we should explore other parameters more. We can try increasing the min_samples_leaf further since increasing it from 1 to 4 improved the performance, and we can also experiment with a different max_features value to see if it affects the loss.*

Longer trajectories

Table: Evaluating performance on a longer trajectory of 30 function calls. GPT-4-Turbo still maintains good performance, suggesting that it is capable of selecting problem-specific hyperparameter settings. The mean rank between all 5 methods (each row + random) is 3.

Model	Beats Random (↑)	Median Change (↑)	Mean Change (↑)	Rank (↓)
GPT-4 Turbo CoT	78.12%	12.10%	18.27 %	2.72
GPT-4 Turbo	90.62%	18.94%	22.34 %	2.22
Bayes Opt$_{RF}$	71.88%	5.91%	14.63 %	3.06
Bayes Opt$_{GP}$	84.38%	11.80%	21.07 %	2.75

We also assess if LLMs can be useful to initialize Bayesian optimization: we find that for a search trajectory of length 30, using the LLM proposed configurations for the first ten steps improves or matches performance on 21 of the 32 tasks (65.6%) for Bayesian optimization.

Code Generation - An Alternative Approach to Tune Hyperparameters

- Given the problem description, we ask the LLM to output the model and optimizer code.
- We search for hyperparameters by allowing the LLM to run its generated code.
- This alleviates the need for human specification of the hyperparameters and their search spaces. It provides a strong initialization and can reduce the initial search
- We experiment on a recent Kaggle challenge to minimize the influence of data leakage into the LLM.

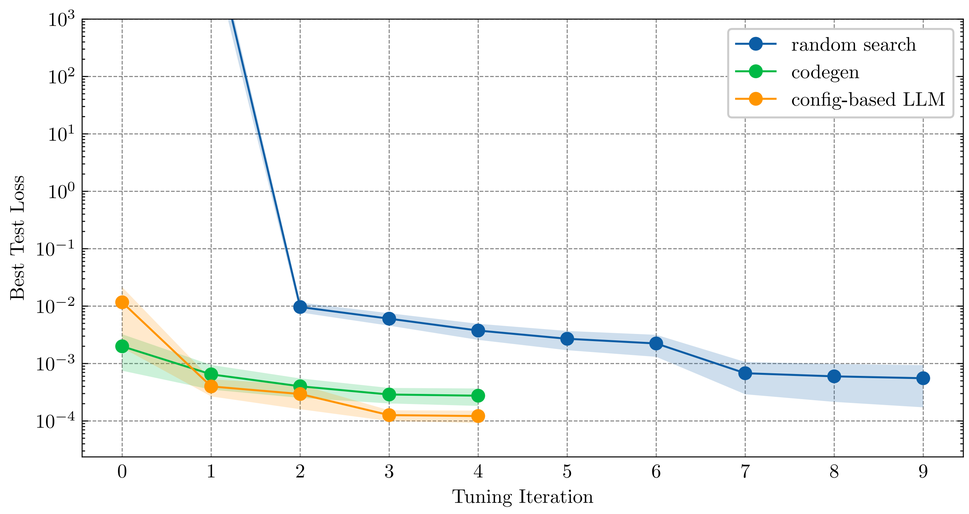


Figure: Test loss on NYC cab Kaggle dataset

2D examples

We consider a set of 2-dimensional toy test functions commonly used in optimization

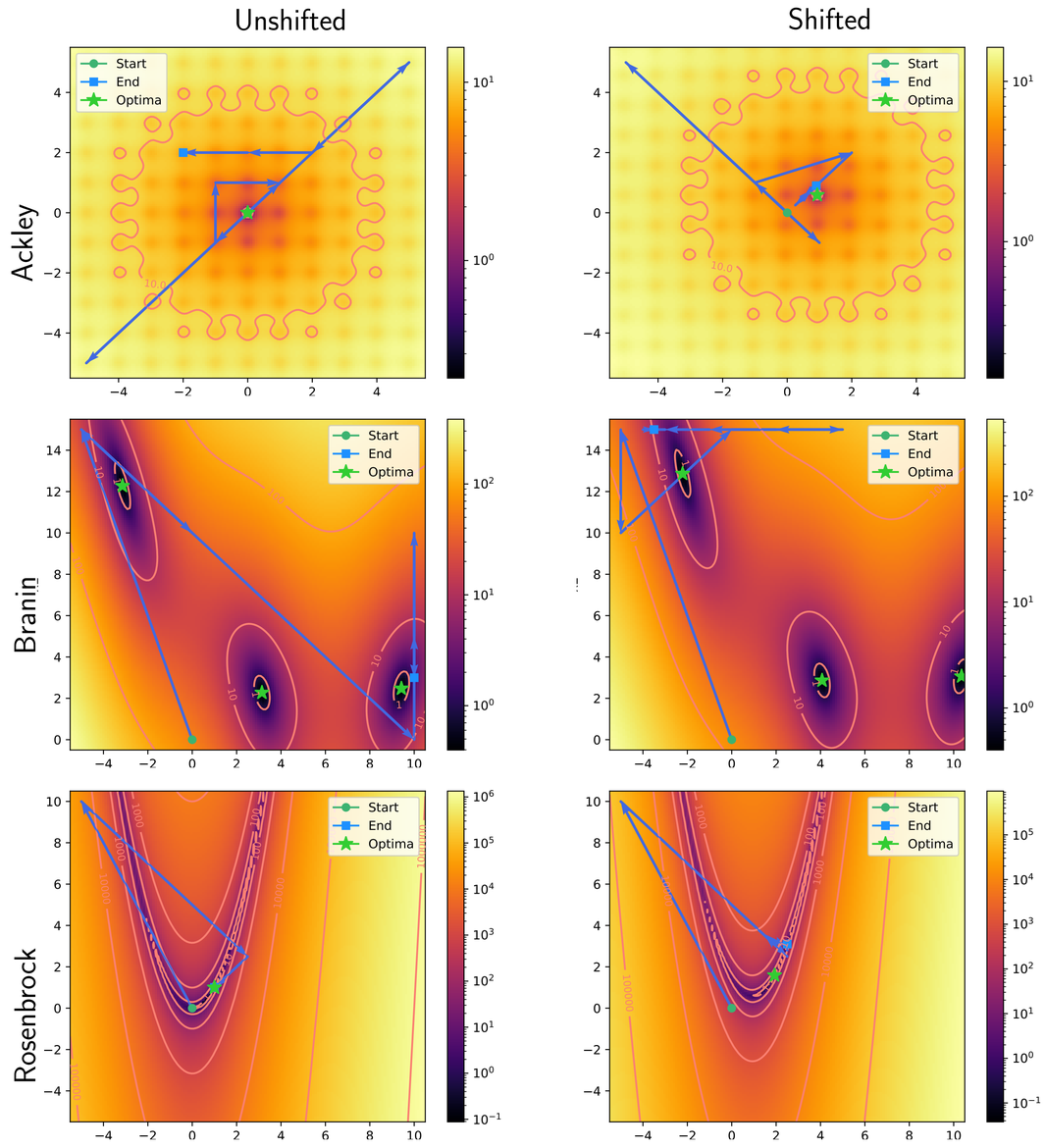


Figure: The LLM optimizer effectively optimizes the function in most situations with few function evaluations, even when we apply random argument shifts to mitigate memorization.

Discussion

- This HPO setting is well-studied, and its dynamic, sequential decision-making nature makes it an intriguing way to evaluate LLMs.
- It is interesting to extend beyond HPO to using LLMs as general assistants for running experiments, as they can offer interactive help to debug and improve models using natural language.