

TDDE19 Advanced Project Course

– AI and Machine Learning

AI Assisted Software Engineering (Part 2)

Mattias Tiger (PhD, AI Researcher)

AI and Integrated Computer Systems (AIICS),
Department of Computer Science

mattias.tiger@liu.se

Recap

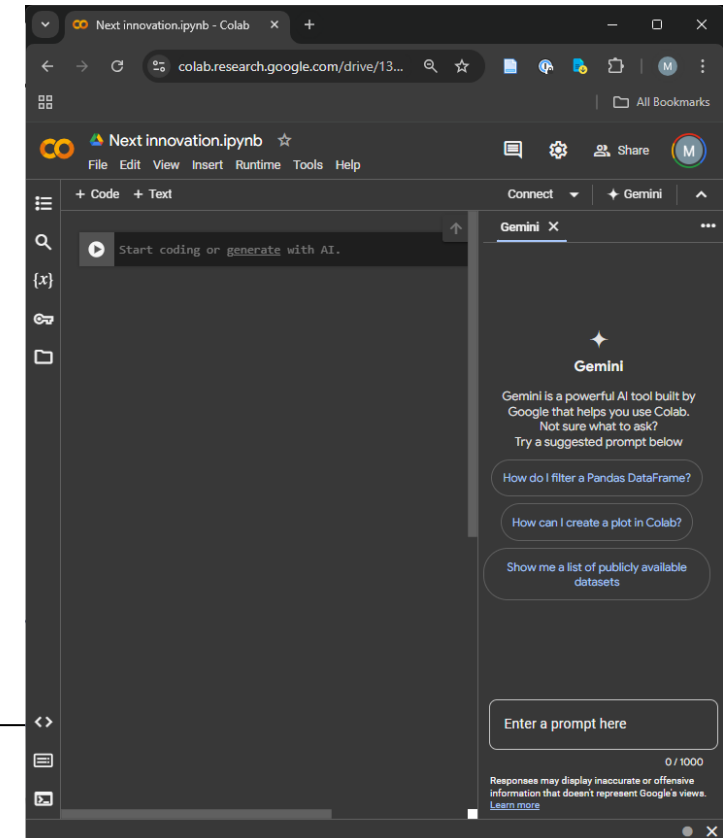
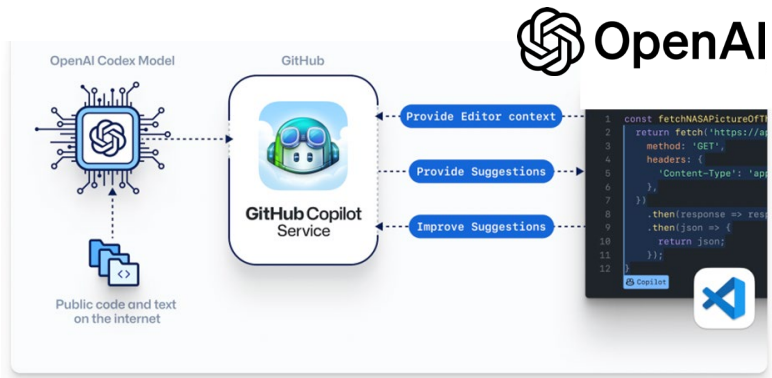
- LLMs are trained on everything and anything the orgs. can get their hands on *"the entire internet"*
- LLMs are instruction-tuned on a lot of different tasks *"all of GitHub"?*  *"everything ever written (digitally)"*
- ❖ But most are 1-step (just like the pretraining data)
- LLMs are RL-trained on test-time roll-outs to solve harder problems using more compute
- ❖ Seemingly multi-step, but very context-dependent.

General rule:

The output must be verified by the user!

Recap | Use-case: LLMs for Programming Assistance

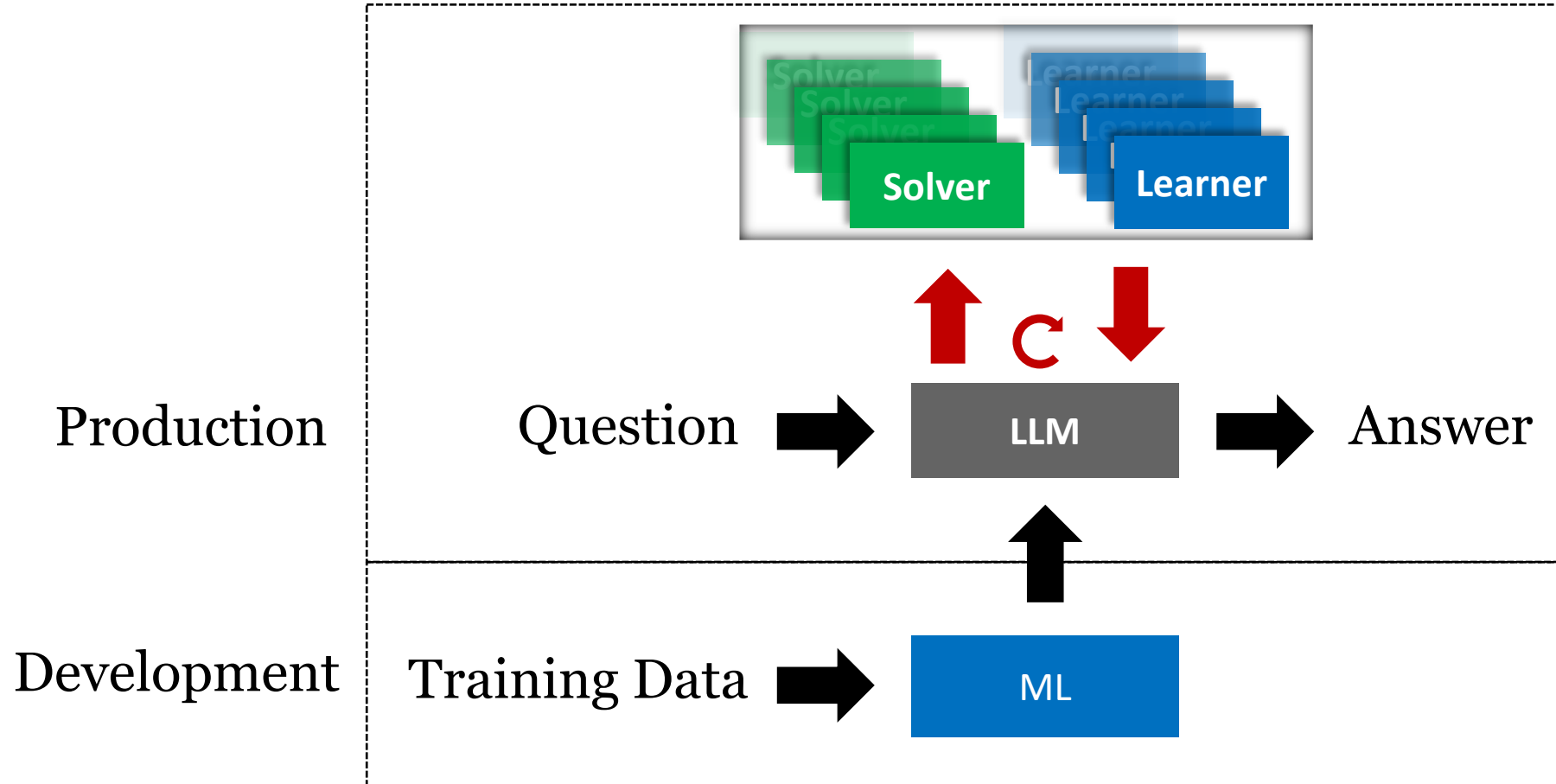
“Today, more than a quarter of all new code at Google is **generated by AI**, then reviewed and accepted by engineers. This helps our engineers do more and move faster.”



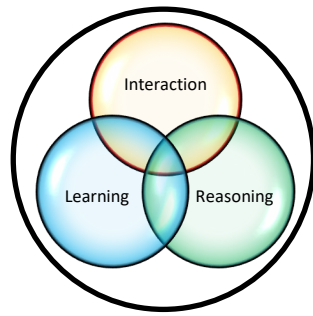
Recap | The Next Big Step for LLMs

- Standardized "tool interface" (MCP) – Natural Language interface for the user.
- Integration of LLMs with **the rest of the AI field**

*Speak in your
native language
with the computer*



Example Application | Parameter Tuning



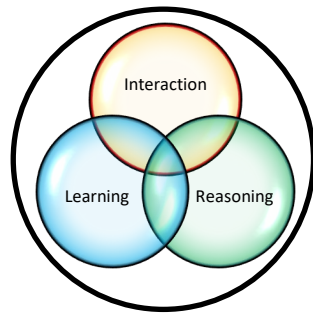
Setting: An objective should be optimized but it is highly expensive (time or cost) to perform each experiments.

Application:

- ML parameter tuning (hyperparameter tuning) ^[1]
- Tuning algorithms to target applications (domain adaptation) ^[2,3]
- Drug discovery, A/B-testing, ...

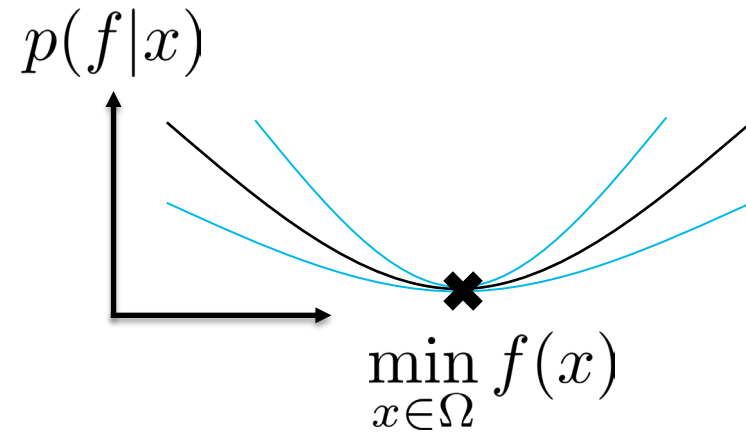
Problem: Search for (global) optima with minimal number of experiments.

Parameter Tuning



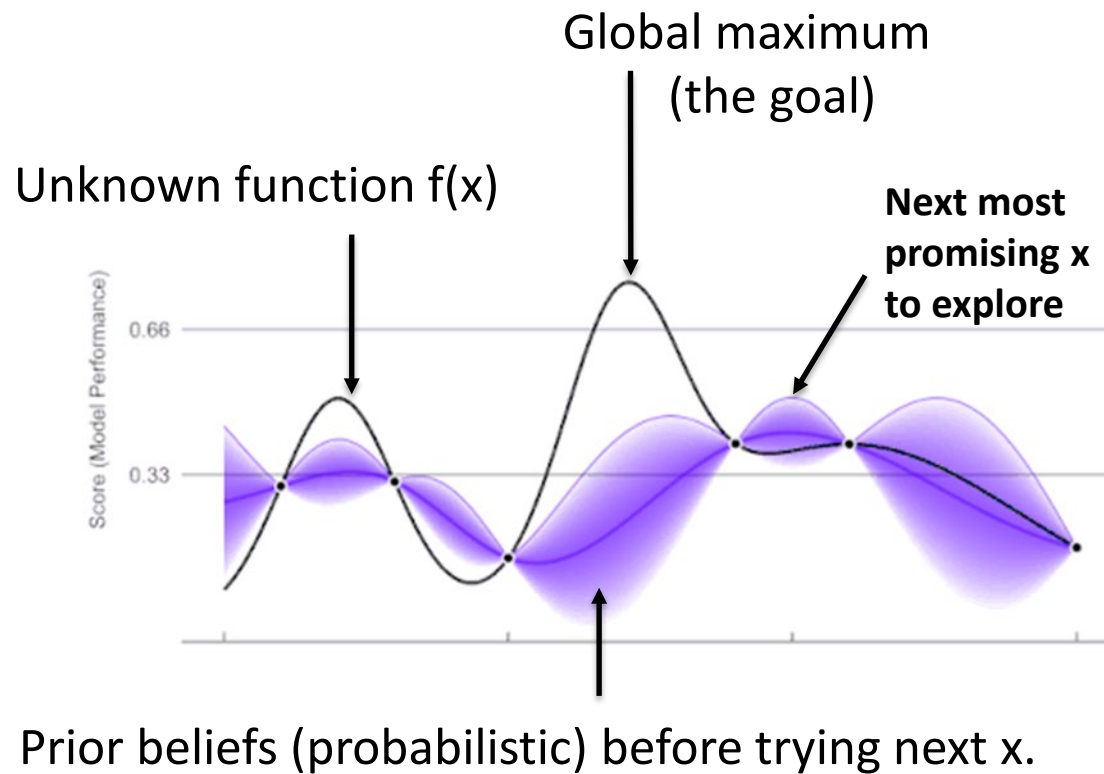
Methods

- **Grid search** (evaluate entire grid and save the best grid point)
- **Random search** (sample a distribution and save the best sample)
- **Gradient-based optimization** (follow the gradient to local minima/maxima)
- **Bayesian Optimization (BO)**
(Update posterior distribution over parameter space after each experiment) [4]

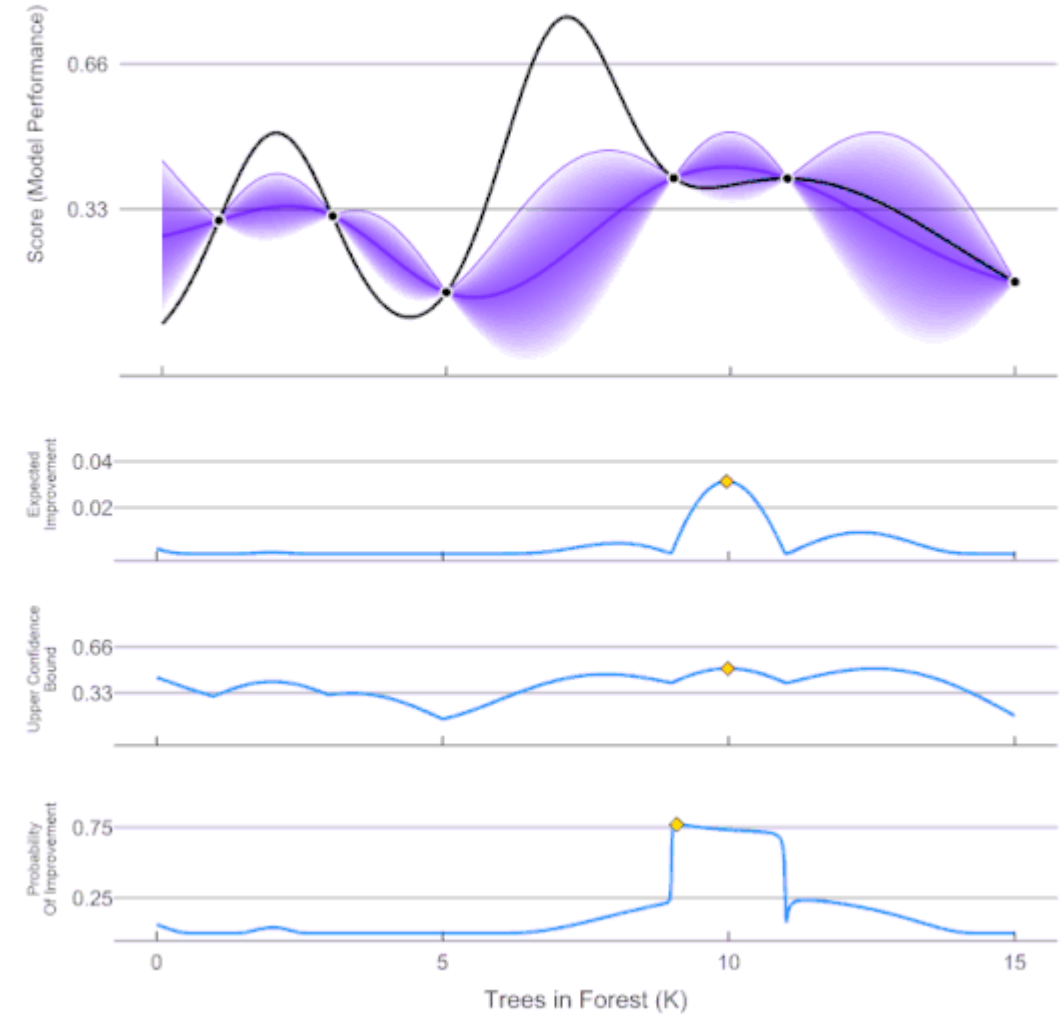


Example | Parameter Tuning

Bayesian Optimization for parameter tuning (example)



ParBayesianOptimization in Action (Round 1)



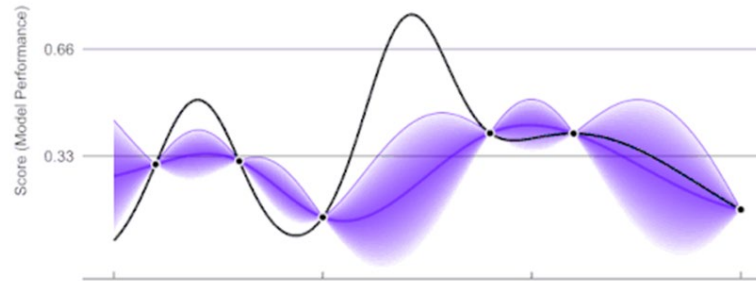
AI Tools | Global Optimization

Bayesian Optimization for parameter tuning (experiment design)

```
1 from hyperopt import fmin, tpe, hp
2 space = hp.uniform(label: 'x', *args: -10, 10)
3 objective = fn(x)
4 best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
```

Search space

What you want to minimize
(structurally unknown)



Number of tries (experiments)
(50-200 often sufficient)

Try it yourself!

objective = **lambda** x: x ** 2

max_evals = 5, 10, 20, 50

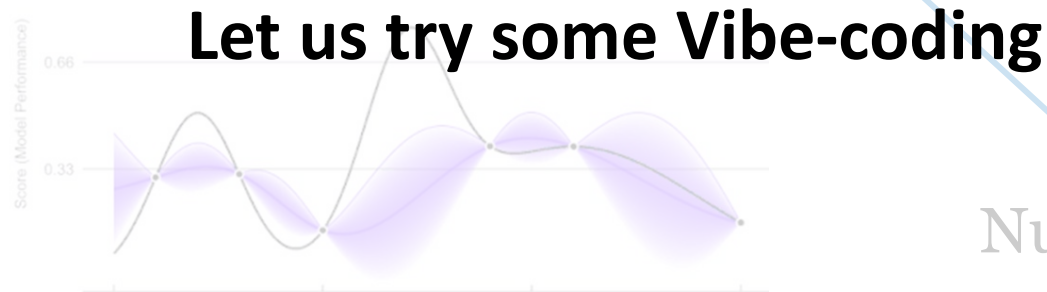
AI Tools | Global Optimization

Bayesian Optimization for parameter tuning (experiment design)

```
1 from hyperopt import fmin, tpe, hp
2 space = hp.uniform( label: 'x',  *args: -10, 10)
3 objective = fn(x)
4 best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
```

Search space

What you want to minimize
(structurally unknown)



Number of tries (experiments)
(50-200 often sufficient)

Try it yourself!

```
objective = lambda x: x ** 2
max_evals = 5, 10, 20, 50
```

Recap

- LLMs are trained on everything and anything the orgs. can get their hands on
- LLMs are instruction-tuned on a lot of different tasks
 - ❖ But most are 1-step (just like the pretraining data)
- LLMs are RL-trained on test-time roll-outs to solve harder problems using more compute
 - ❖ Seemingly multi-step, but very context-dependent.

"the entire internet"

*"everything
ever written
(digitally)"*

"all of GitHub"?



General rule:

The output must be verified by the user!

➤ In AI Assisted Software Development

- **You** have to **provide** the **context**
- **You** have to **mitigate** compounding **errors** (hallucinations and error-rates)
- **You** have to **know** (or figure out) **what to ask for**

LLM Interaction – Best Practice

- **Don't ask a question without *first building coherent context to the question***
 - De-ambiguate and focus the answer (home in on the right parts of “everything ever written”).
 1. Supply best practices of task X, guidelines for X, API for X, etc.
 2. Ask it to give a detailed summary of 1. focused on a general description of your task
 3. Give examples of what you want (e.g. code or documents) to use as “style transfer”
 4. Ask the actual question (what you want the LLM to do/solve).
- **Ask the LLM to describe/summarize what is good to know or needed to solve a task. Repeatedly ask the LLM for weaknesses or critical details missing.**
 - LLM-driven Prompt refinement
- **Provide a dump of what you want to achieve.**
Then ask the LLM to ask you questions until it can convince you that it has “understood” and has specified everything important to know.
 - Let the LLM guide you in specifying the unknown unknowns

LLM Interaction – Best Practice

- **Encourage for de-ambiguation with any request:**
 - “Let me know if you have any questions before we start.” (E.g. ChatGPT Deep Research)
- **Re-use lessons learns from chat history**
 - “Based on our chat history, create a document which capture common mistakes and how to avoid them or solve them.”
 - “Summarize difficulties we have had in understanding each other.”
- **Guidance/planning rather than editing**
 - “Provide me a list of suggested changes. Do not update any files.”
- **Encourage progress tracking**
 - “Create a progress tracking document and keep it up-to-date in-between every new feature. Make sure that it has verification steps and outline development best-practices.”
- **Automated prompt engineering (transfer to new LLM instance)**
 - Write a prompt for X. Assume it is for an LLM with no previous knowledge about our chat history.

LLM Coding – Best Practice

- **Dump (i.e. train of thoughts) what you want to build. Add as much as you can think about.**
- **Let the LLM guide you to expand and make choices:**
 - “Your mission is to assist me in writing necessary technical documents that describe and enable completion of this software project. Start with iteratively asking me questions to clarify all relevant aspects of the project.”
- **Create technical documents:**
 - “Create markdown files of a) requirements specification, 2) high-level project plan and 3) a readme file. Make sure that the project plan contain progress tacking and verification steps. The documents will be used by an LLM to complete the project with minimal human supervision. Do verification tests on your own as much as possible before involving the human user for final verification, at any point during dev.”
 - “I want you to: 1) Break down the problem into different phases. 2) Create a detailed plan (markdown file) for each phase. 3) Make sure that all technical documents refer to each other and that all technical documents, and their intended usage, is clearly described in the readme file.”
- **Use additional scratch-pads for meta-progress and of-tangents (e.g. debugging or vision-changes)**
 - “Create a memory.md and a progress.md which we can include often in context. They should together capture essential things to keep in mind for the current task at hand, as well as general progress tracking.”

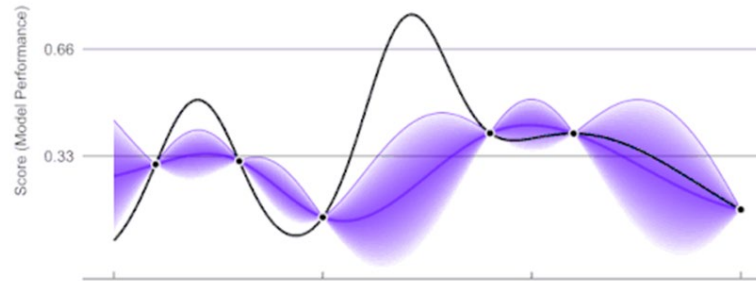
AI Tools | Global Optimization

Bayesian Optimization for parameter tuning (experiment design)

```
1 from hyperopt import fmin, tpe, hp
2 space = hp.uniform( label: 'x', *args: -10, 10)
3 objective = fn(x)
4 best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
```

Search space

What you want to minimize
(structurally unknown)



Number of tries (experiments)
(50-200 often sufficient)

Try it yourself!

objective = **lambda** x: x ** 2

max_evals = 5, 10, 20, 50

AI Tools | Global Optimization

Bayesian Optimization for parameter tuning (experiment design)

```
1 from hyperopt import fmin, tpe, hp
2 space = hp.uniform('x', -10, 10)
3 objective = fn(x)
4 best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
```

Search space

What you want to minimize
(structurally unknown)



Number of tries (experiments)
(50-200 often sufficient)

Try it yourself!

```
objective = lambda x: x ** 2
max_evals = 5, 10, 20, 50
```

New frontiers of software engineering?

Previous paradigm

- Specialize by memorizing and practicing using specific techniques (A*, BO, YOLO).
- Break down problems to solve them piece by piece.
- Divide your code into small increments or features that are easy to grasp and explain to colleagues.
- Scrum-ish, since thinking it through in detail is infeasible and beyond our ability.
- Leverage past experience and reference materials (e.g. manuals or blogs).

New paradigm?

- Generalize by knowing what you can ask for: What areas are previously explored, and what algorithms exist. Specialize in QA.
- Break down problems so that the LLM's context does not get overwhelmed.
- Divide your code into separate goal-focused conversations. Global changes that can be hard to grasp. (Hide goal?)
- Repeated “Waterfall”, since more context makes the implementation more aligned.
- Approximate context retrieval and interpolation of (almost) all code on Github.

Mattias Tiger

AI och Integrerade Datorsystem (AIICS),
Institutionen för Datavetenskap

www.ida.liu.se/~matti23/mattisite/research/

www.liu.se/ai-academy

www.liu.se/medarbetare/matti23