# Grassland: A Rapid Algebraic Modeling System for Million-variable Optimization

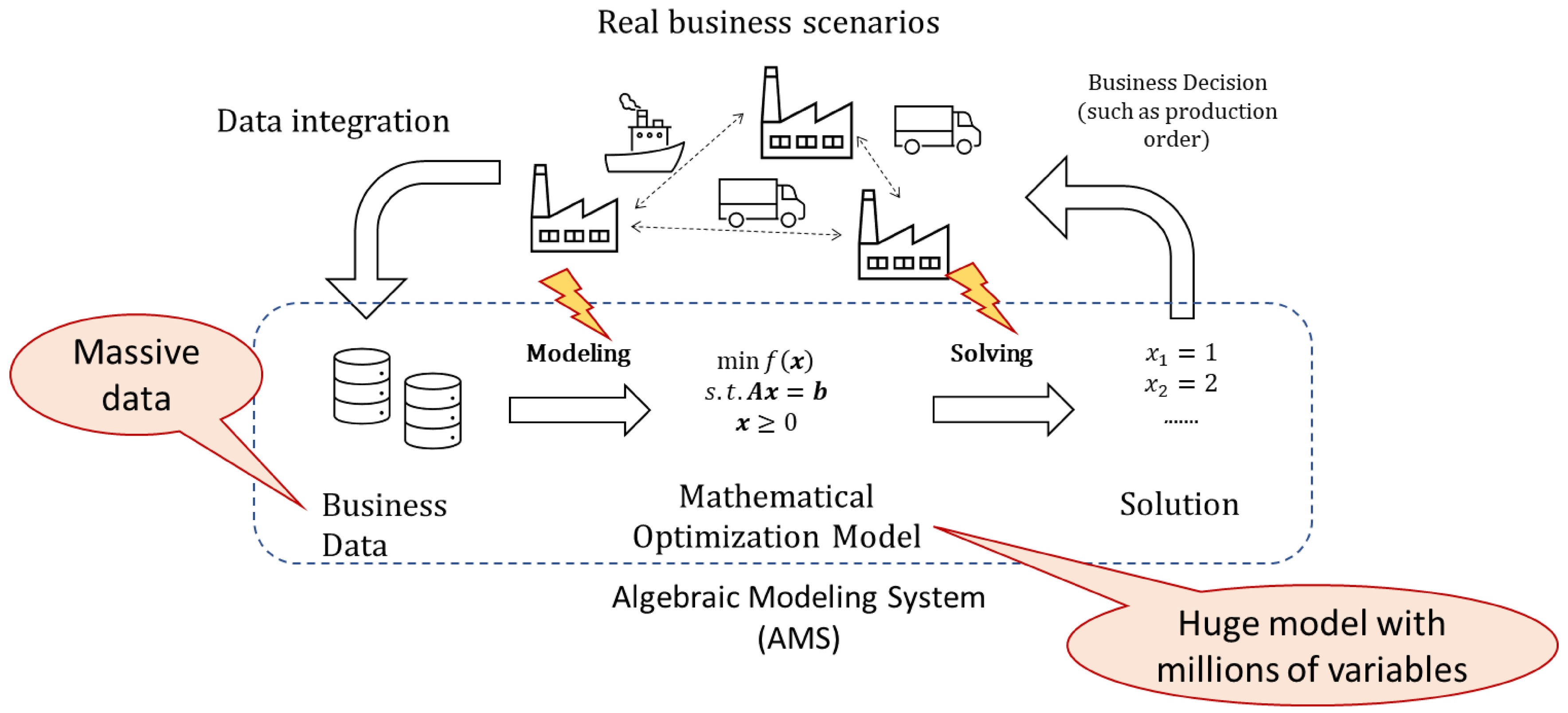
Xihan Li[1], Xiongwei Han[2], Zhishuo Zhou[3], Mingxuan Yuan[2] , Jia Zeng[2] and Jun Wang[1]

[1]University College London [2]Huawei Noah's Ark Lab [3]Fudan University

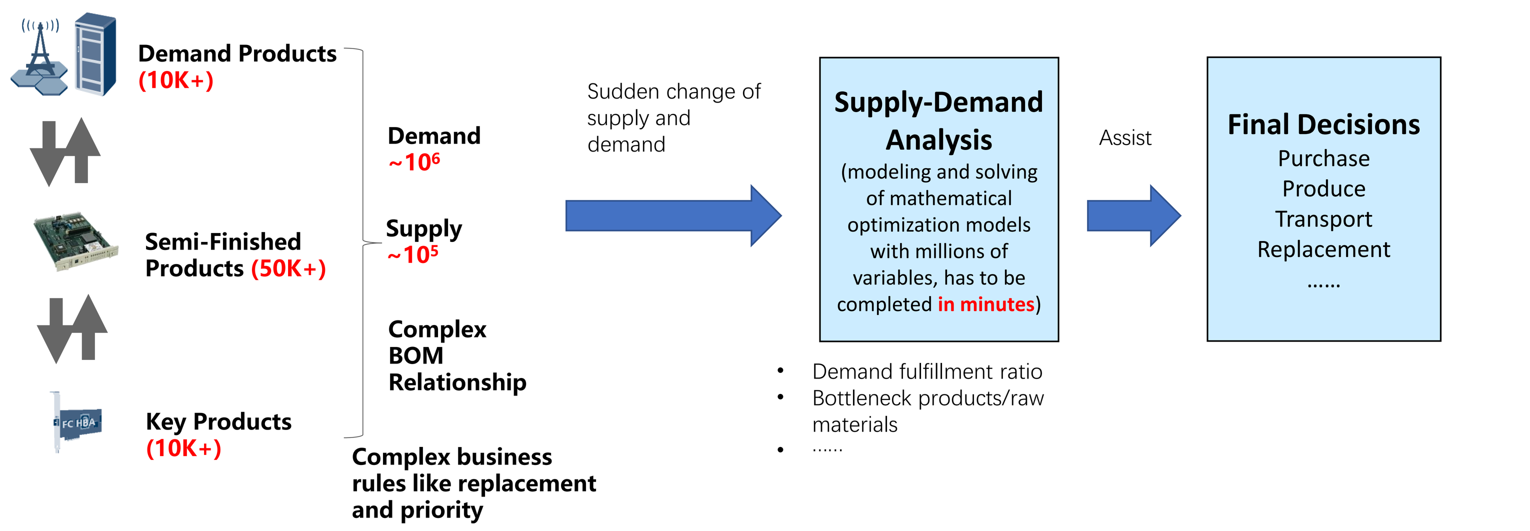
arXiv: https://arxiv.org/abs/2108.04586

## Introduction

Mathematical optimization methods have extensive and profound applications in manufacturing, transportation, logistics, energy, finance and many other fields.
Mathematical optimization = **Modeling** + **Solving**
With bursting scale of data today, the business model can have **millions of variables**, both modeling and solving can take **hours** – desperately slow!



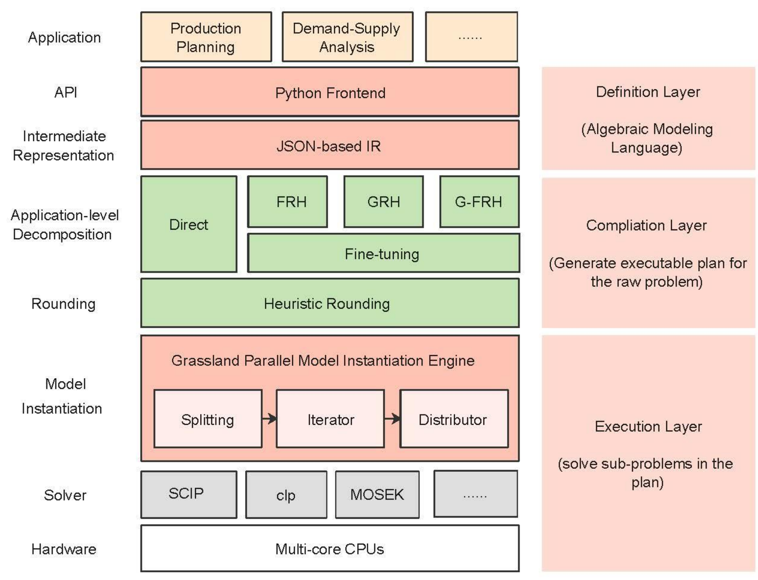

But in real business, we need **rapid optimization** to adapt to highly dynamic market changes – every minute counts! Especially for **Huawei's supply chain** – we need to do modeling and solving of million-variable models **in minutes.**




To achieve this, we developed **Grassland**, a rapid algebraic modeling system to accelerate both the modeling and solving stage of mathematical optimization.

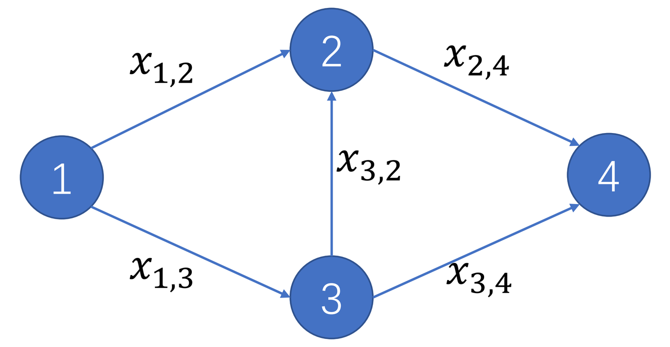
It can model and solve **million-variable** business models in **3~5 minutes**!

Grassland will be integrated as the default modeling system of **Huawei OptVerse AI Solver.**




## Efficient Modeling: A Fast Algorithm to Instantiate Millions of Linear Constraints

A simple mathematical optimization example:
balance constraint of network flow model



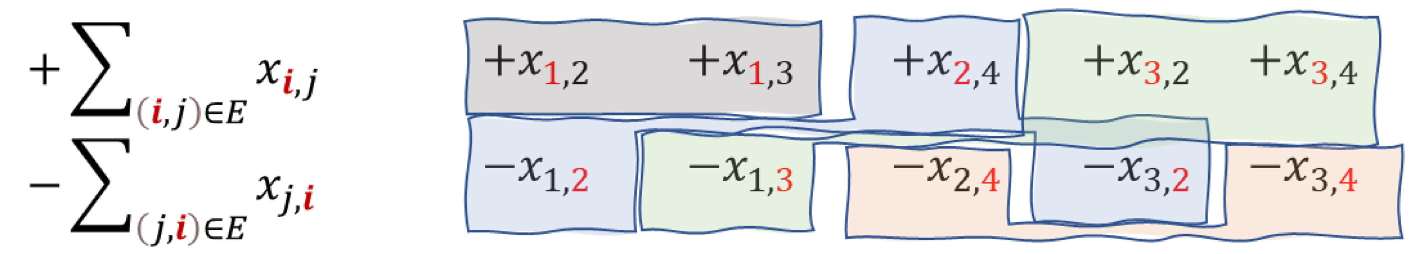

$$V = \{1,2,3,4\}$$
$$E = \begin{bmatrix}(1,2)\\(1,3)\\(2,4)\\(3,2)\\(3,4)\end{bmatrix}$$
$$S = [1,0,0,-1]$$

$$\text{constraint i: } \sum_{(i,j)\in E} x_{i,j} - \sum_{(j,i)\in E} x_{j,i} = s_i, \forall i \in V$$

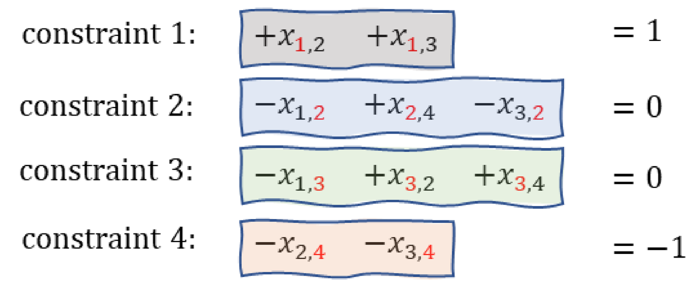
- constraint 1: $+x_{1,2} + x_{1,3} = 1$
- constraint 2: $+x_{2,4} - x_{1,2} - x_{3,2} = 0$
- constraint 3: $+x_{3,2} + x_{3,4} - x_{1,3} = 0$
- constraint 4: $-x_{2,4} - x_{3,4} = -1$

Generate constraints line by line?
Time complexity $O(|V||E|)$
Too slow! Won't work with millions of vertices and edges.

New approach:
1. Generate all terms in batches

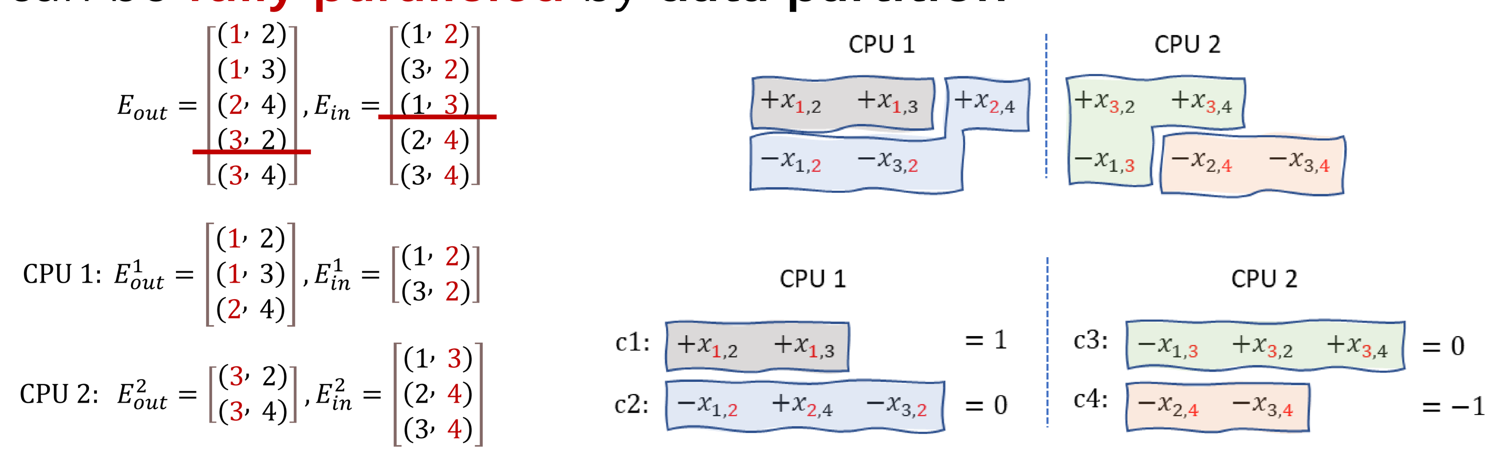



2. Distribute all terms to their corresponding constraints

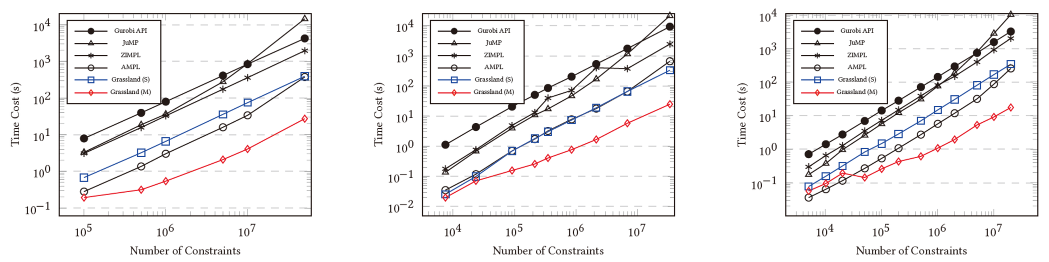



$O(|E|)$ time complexity!

It can be **fully paralleled** by **data partition**




| | P-Median | Offshore Wind Farming | Food Manufacture I |
|---|---|---|---|
| Gurobi Py API | 410.20 | 533.71 | 744.39 |
| JuMP | 278.08 | 169.08 | 789.86 |
| ZIMPL | 174.00 | 400.47 | 399.16 |
| AMPL | 15.94 | 17.71 | 31.65 |
| Grassland (S) | 35.91 | 18.85 | 80.83 |
| Grassland (M) | **2.09** | **1.67** | **5.28** |

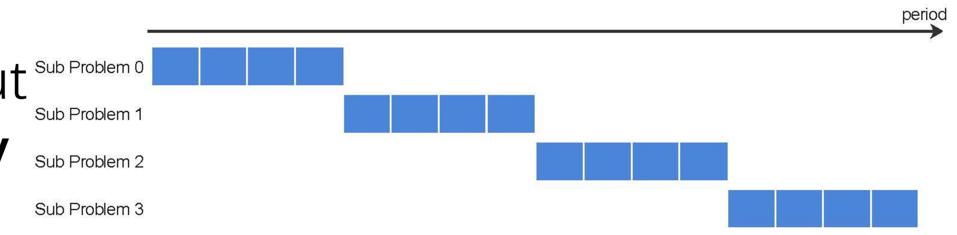


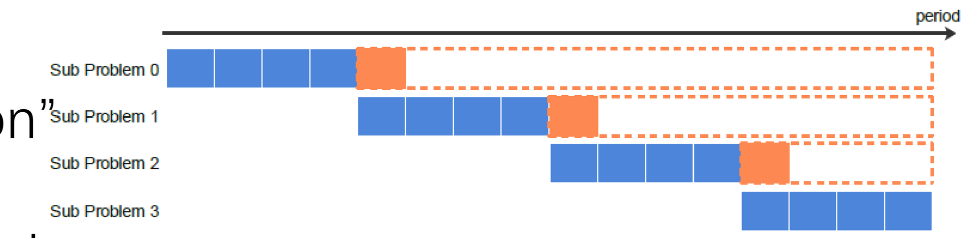

**Experiment: 6-10x speedup** over leading commercial modeling software. Instantiate million-variable models **in seconds**!

## Efficient Solving: Sequential Decomposition for Large-Scale Optimization

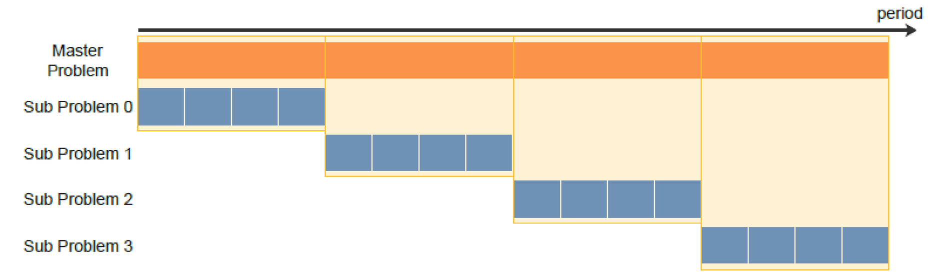
Why business optimization models are always so large? An important reason is that they need **sequential, dynamic decision making**. E.g., when we have 10K decision variable in a single period, we will need 10K * 100 = 1M variables for 100 periods.

- **Rolling Horizon (RH):** partition the periods, but the decision will be **very short-sighted**. Global optimality will be lost.
- **Forward RH:** Add "aggregated information" to the end of the sub-model to improve global optimality.
- **Guided RH**: Solve a "master problem", then Add a soft constraint that the solution of sub-problems should be aligned with the master problem.
- **Guided FRH**: Forward RH + Guided RH.
- **Fine-Tuning:** Re-optimize the variables in first several periods to improve global optimality.

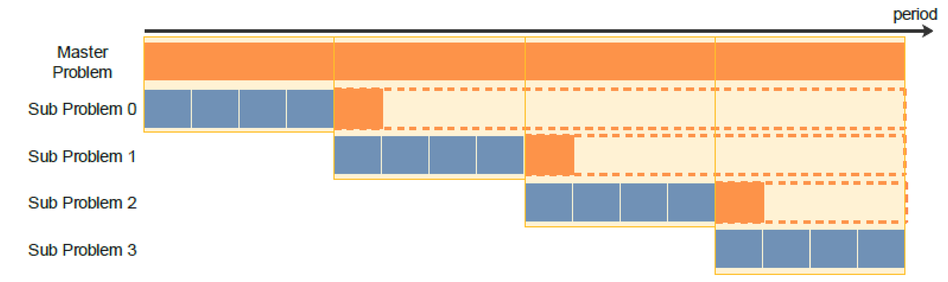


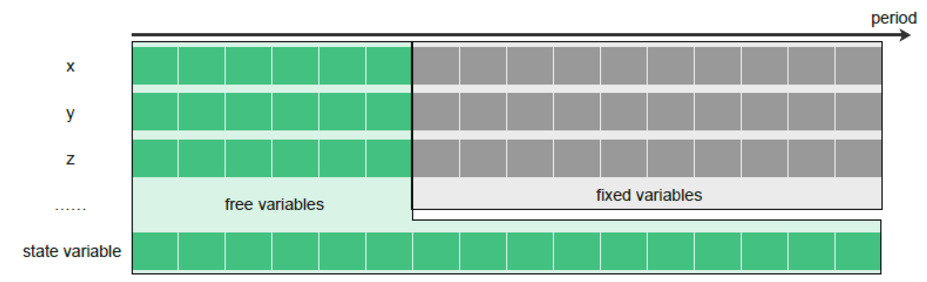

(a) Forward RH (FRH), [8]

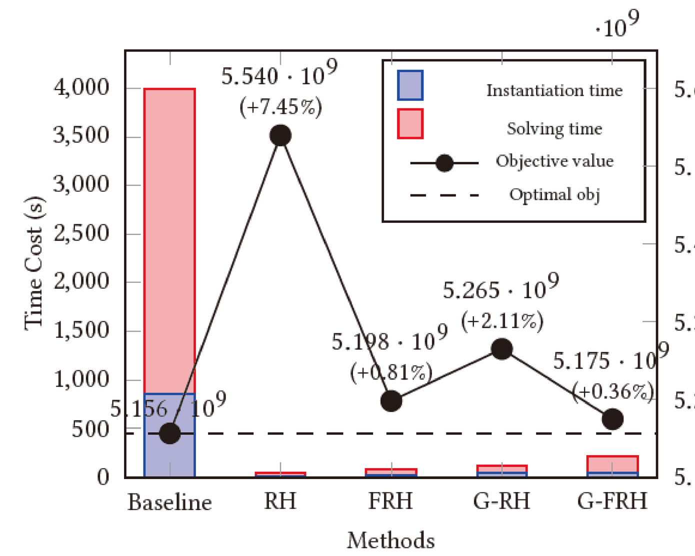



(b) Guided RH

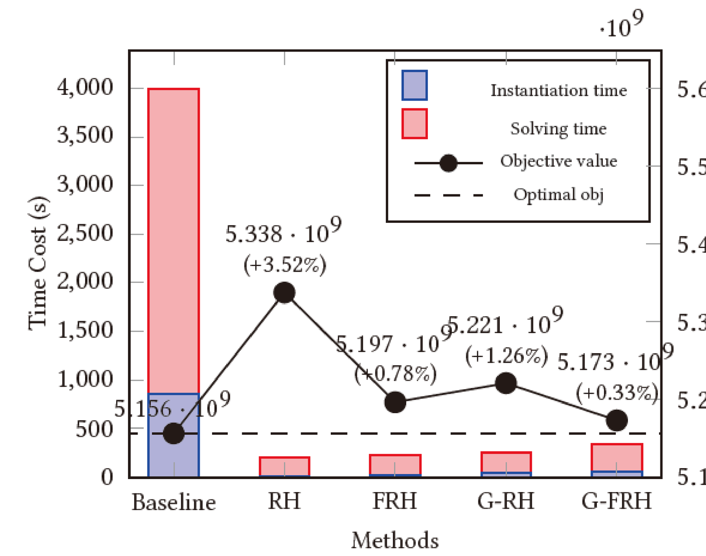



(c) Guided FRH




**Experiment: 20x speedup** (4000s → 200s) with optimality loss of only **3.6‰**




The optimality loss can be further reduced to **3.3‰** with fine-tuning