Black or White? How to Develop an AutoTuner for Memory-based Analytics

Mayuresh Kunjir and Shivnath Babu

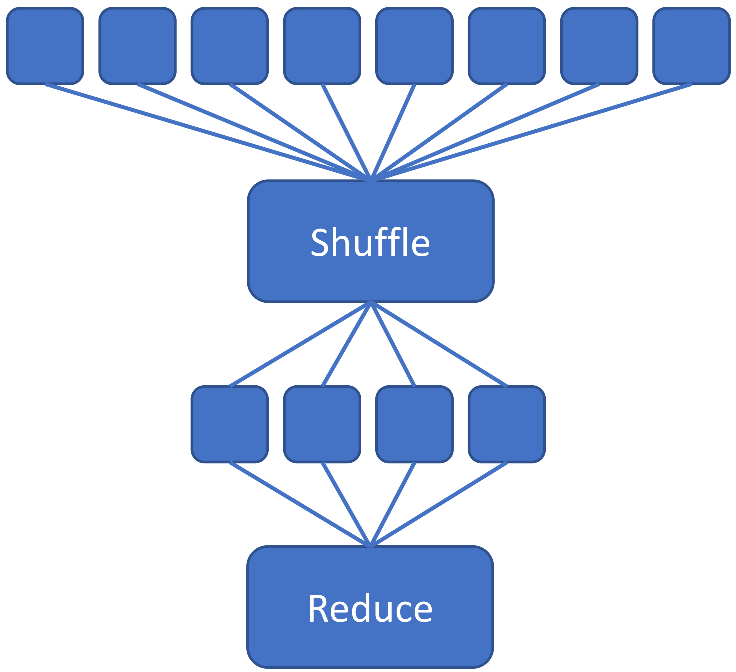
Review by Ross Tooley

Contents

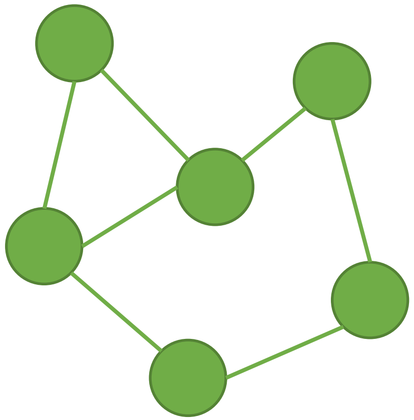
Background

Apache Spark

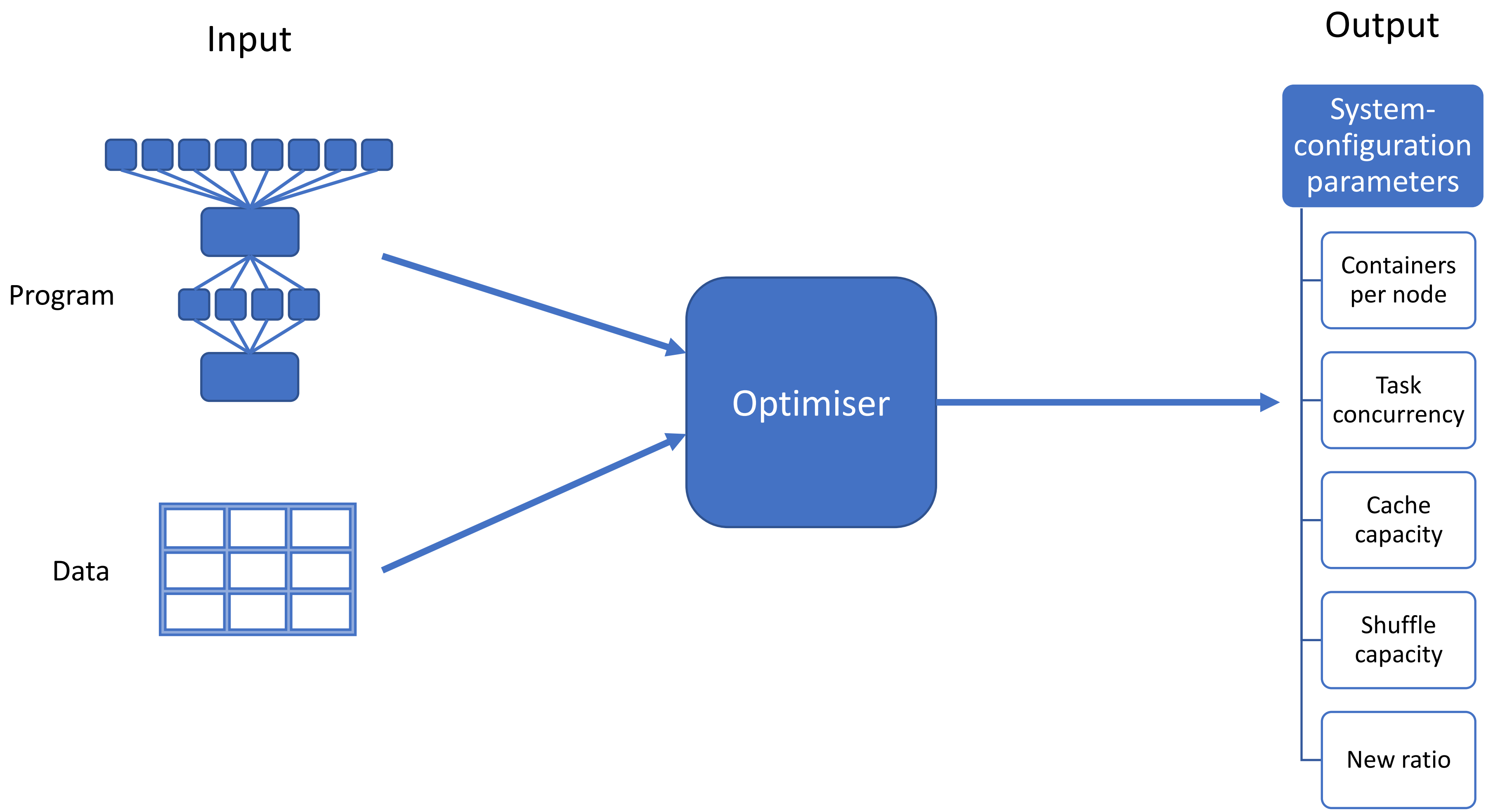
Bulk-synchronous parallel



Distributed on cluster



The optimisation problem



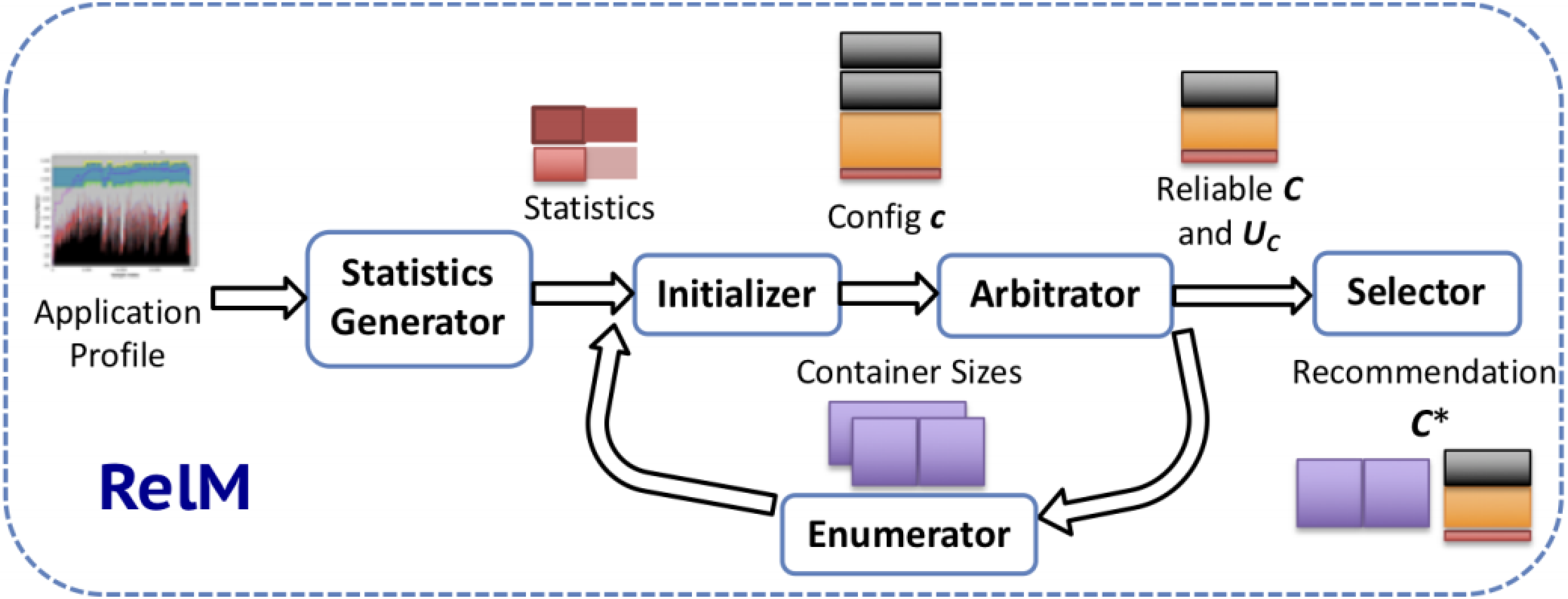
Existing Spark optimisers

- Guidelines for manual tuning
  - Spark
- Pure BO
  - Fekry et al.
- Genetic algorithms
  - Yu et al.
- Regression trees
  - Wang et al.
- And more…

Paper overview

RelM

Design



Statistics
Config *c*
Reliable *C*
and U_c
Application
Profile
Statistics
Generator
Initializer
Arbitrator
Selector
Container Sizes
Recommendation
c^*
RelM
Enumerator

Components

Statistics

Notation	Description	Example
N	Containers per Node	1
M_h	Heap size	4404MB
CPU_{avg}	Average CPU usage	35%
$Disk_{avg}$	Average disk usage	2%
M_i	Code Overhead 90%ile value	115MB
M_c	Cache Storage 90%ile value	2300MB
M_s	Task Shuffle 90%ile value	0MB
M_u	Task Unmanaged 90%ile value	770MB
P	Task Concurrency	2
H	Cache Hit Ratio (the fraction of cached data partitions actually read from cache)	0.3
S	Data Spillage Fraction (the fraction of shuffle data spilled to disk)	0

Enumerator

	Task concurrency			
# containers	(1,1)	(1,2)	(1,3)	...
	(2,1)	$\ddots$		
	(3,1)			
	$\vdots$			

Initialiser

$$m_c = m_h * \min\left(\frac{M_c}{H * M_h}, 1 - \delta\right)$$

$$m_s = \min\left(\frac{M_s}{1 - S/P}, (1 - \delta) * m_h\right)$$

$$NR = \text{ceil}\left(\frac{M_i + m_c}{m_h - M_i - m_c}\right)$$

$$m_o = m_h * \frac{NR}{NR + 1}, m_e = m_h * \frac{1}{NR + 1} * \frac{SR - 2}{SR}$$

$$p^{CPU} = \frac{1}{n}\frac{(1 - \delta) * 100}{CPU_{avg}/P}, p^{disk} = \frac{1}{n}\frac{(1 - \delta) * 100}{Disk_{avg}/P}$$

$$p^{mem} = \frac{(1 - \delta) * m_h}{M_u}, p = \min(p^{CPU}, p^{disk}, p^{mem})$$

Arbitrator

Algorithm 1 RelM Arbitrator

Input: Configuration $\mathbf{c} = (M_i, M_u, p, m_c, m_s)$, Safety factor δ

if $(M_i + M_u) > (1 - \delta) * m_h$ **then**
  Return flagging insufficient memory
end if
while $(M_i + p * M_u + m_c) > m_o$ **do**
  one of the following three in a round-robin manner:
  I. Decrease p by 1 if $p > 1$
  II. Reduce m_c by M_u ensuring that $m_c > 0$.
    Change GC pools using Equation 3.
  III. Increase m_o by M_u ensuring $m_o < (1 - \delta) * m_h$
end while
Set shuffle memory $m_s = \min(m_s, 0.5 * m_e/p)$
Set output $\mathbf{C} = (M_i, M_u, p, m_c, m_s)$

Selector

$$U_C = \frac{M_i + m_c + p * (M_u + m_s)}{m_h}$$

Bayesian Optimisation and Reinforcement Learning

What’s the idea here?

1

$$q_1^{\mathbf{x}} = \frac{M_i + \min(m_c^{\mathbf{x}}, m_c) + p^{\mathbf{x}} * (M_u + \min(m_s^{\mathbf{x}}, m_s))}{m_h^{\mathbf{x}}}$$

$$q_2^{\mathbf{x}} = \frac{M_i + m_c}{\min(m_o^{\mathbf{x}}, m_c^{\mathbf{x}})}, \quad q_3^{\mathbf{x}} = \frac{p^{\mathbf{x}} * \min(m_s^{\mathbf{x}}, m_s)}{0.5 * m_e^{\mathbf{x}}}$$

Quasi-parameters

$$\mathbf{q^x} = \{q_1^{\mathbf{x}}, q_2^{\mathbf{x}}, q_3^{\mathbf{x}}\}$$

2

Extend input space with q

$$\{x_1, x_2, x_3, x_4, x_5, q_1^{\mathbf{x}}, q_2^{\mathbf{x}}, q_3^{\mathbf{x}}\}$$

3

Use RelM to predict quasi-parameters for each real configuration

Bayesian Optimisation

Gaussian Process (Model)

$$GP(\mathbf{x}, y)$$

$$\Big\downarrow$$

$$GP(\mathbf{x} \cup \mathbf{q}, y)$$

Comparison to BOAT

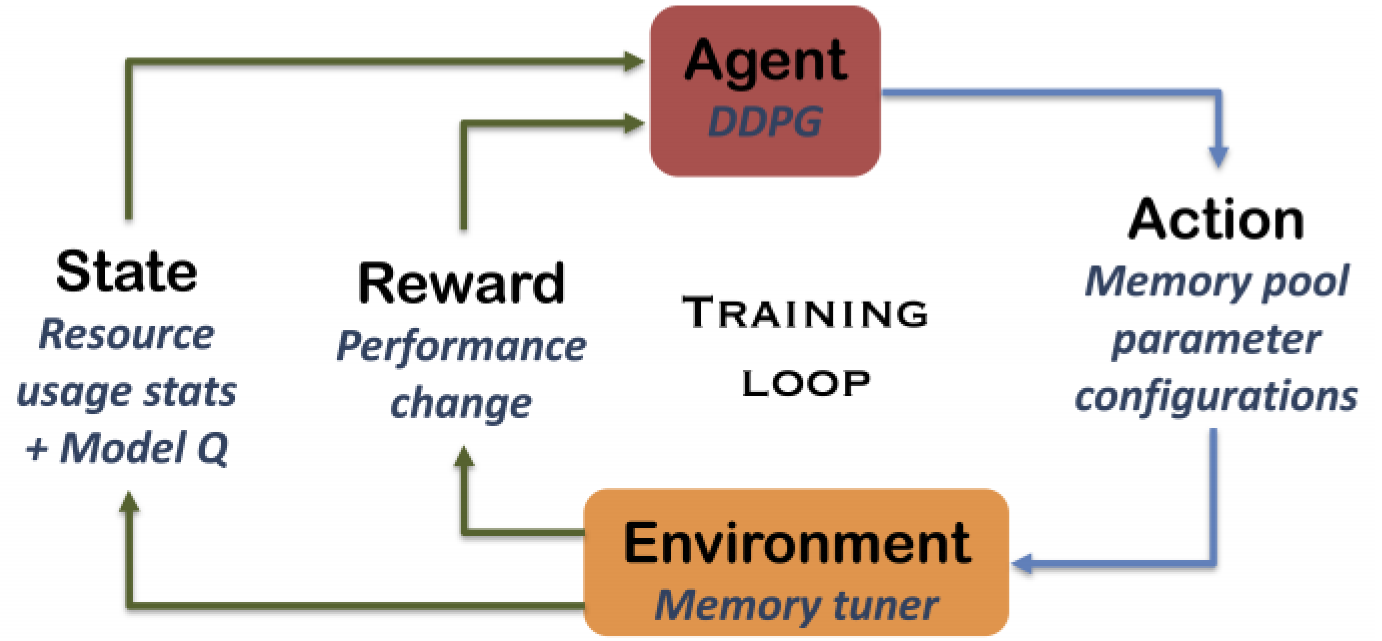
Kunjir & Babu

$$GP(\mathbf{x} \cup \mathbf{q}, y)$$

BOAT

$$\begin{tikzcd}
GP(\mathbf{x}, q_1) \arrow[dr] & GP(\mathbf{x}, q_2) \arrow[d] & GP(\mathbf{x}, q_3) \arrow[dl] \\
 & GP(\mathbf{q}, y) &
\end{tikzcd}$$

Reinforcement Learning (DDPG)



Results

Performance improvement

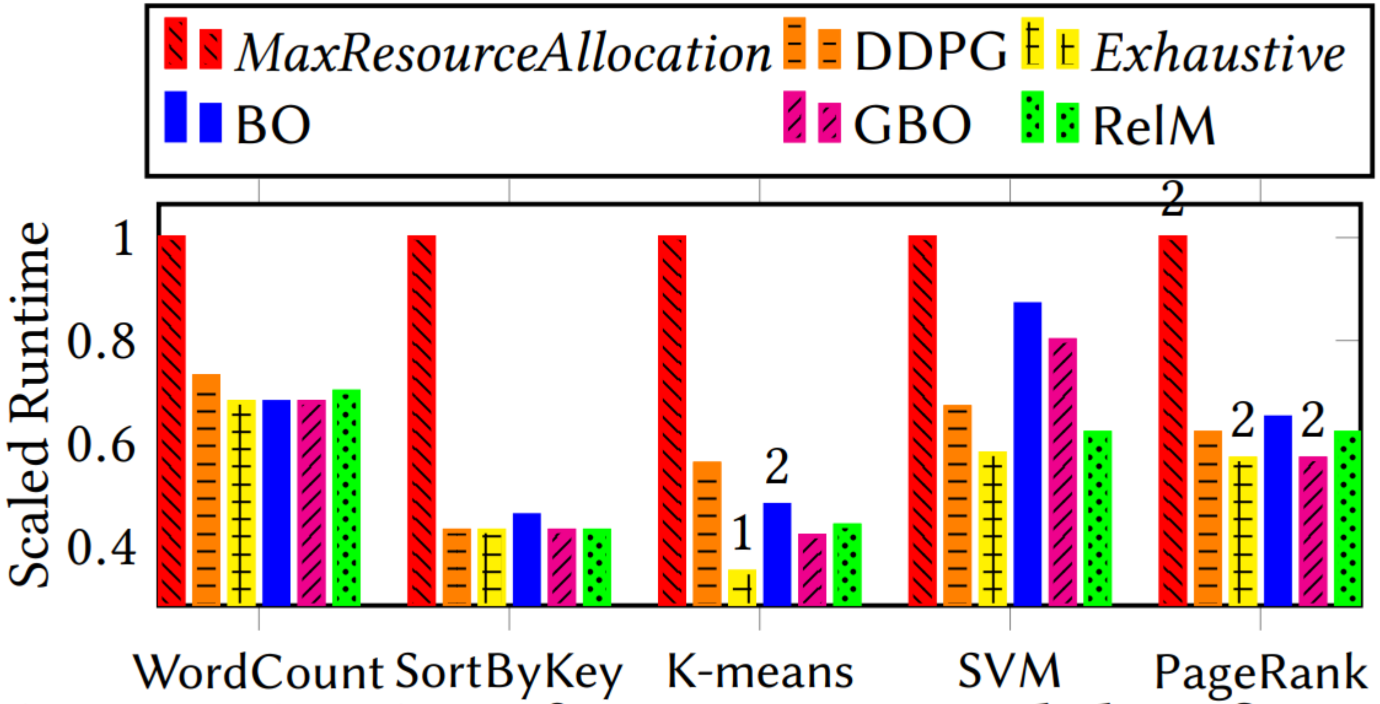


Figure 15: Runtime of every recommended configuration is scaled to the runtime of *MaxResourceAllocation*. Number of failed containers is shown on top of bars.

Convergence time

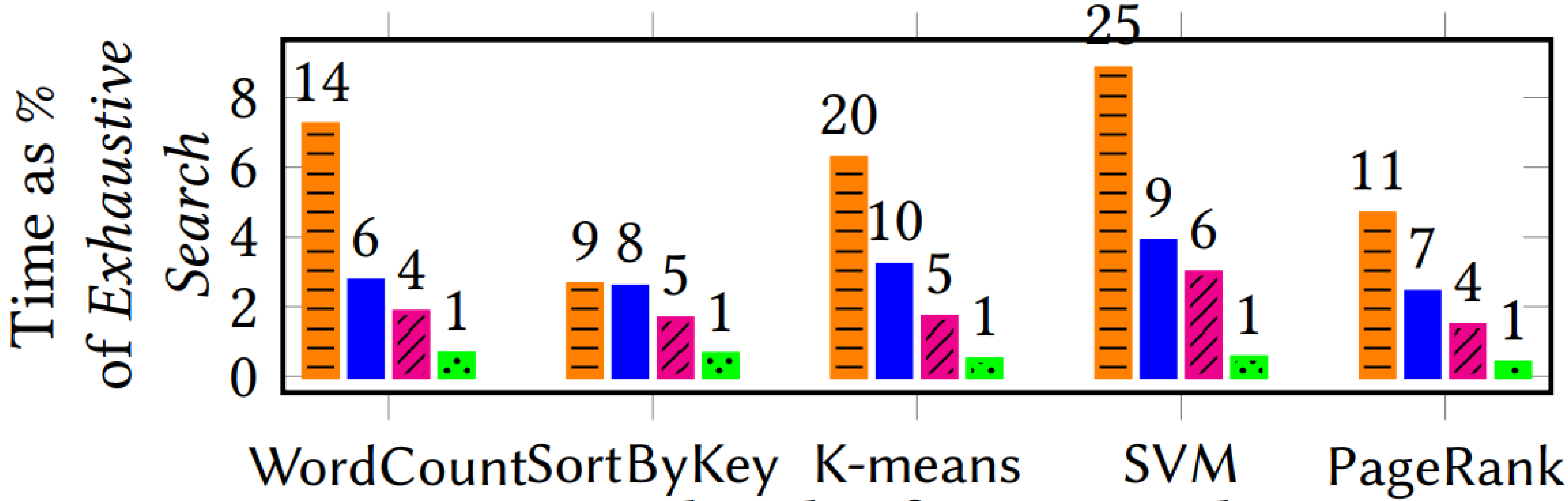


Figure 14: Training overheads of tuning policies. Number of iterations is shown on top of bars.

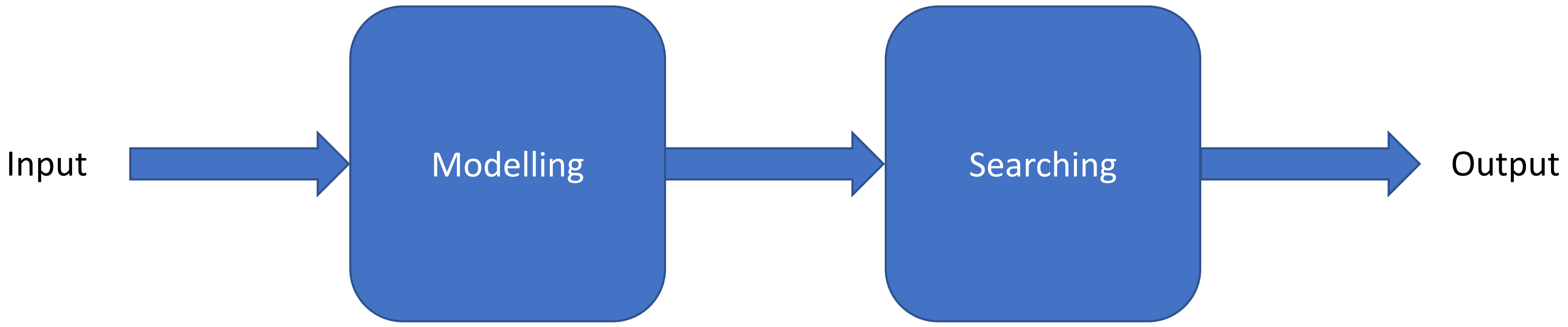
Looks good… any issues?

Overfitting

Comparisons

Comparing all optimisers

The general structure



Contrasting search methods

- **Bayesian Optimisation**
  - BOAT
  - Kunjir & Babu
  - CherryPick
- **Reinforcement Learning**
  - DDPG
    - QTune
    - Kunjir & Babu
  - REINFORCE
    - Mirhoseini et al.
  - Thompson Sampling
    - Bao
- **Genetic**
  - REGAL
- **Population-Based Training**
  - Jaderberg et al.
- **Back-tracking**
  - TASO

Contrasting model methods