Predict and Conquer: Navigating Algorithm Trade-offs with Quantum Design Automation

Simon Thelen[1,2] **and Wolfgang Mauerer**[1,3]

QCE2025, Q-STAV 2026
February 23, 2026

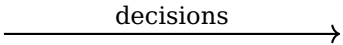
[1] Technical University of Applied Scienes Regensburg, [2] University of the Bundeswehr Munich, [3] Siemens AG, Technology

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

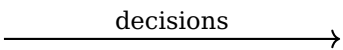
Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...



Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...



Quantum algorithms

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

Quantum algorithms

Example use case: combinatorial optimisation

Application requirements
- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

Quantum algorithms
- Quantum annealing
- VQE
- QAOA
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

← influence

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions →

← **predict** influence

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

decisions

predict
influence

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

automate decisions →

← **predict** influence

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

Example use case: combinatorial optimisation

Application requirements

- Solution quality
- Runtime
- # Qubits
- Noise resistance
- ...

automate decisions →

← **predict** influence

Quantum algorithms

- Quantum annealing
- VQE
- QAOA
  - Standard QAOA
  - Warm-starting QAOA
    - # Iterations
    - # Layers
    - Problem formulation
    - ...
  - Recursive QAOA
  - ...
- ...

⇒ **Software framework**

Simulations

Ideal

Noisy

Simulations

Ideal

Noisy

Metrics

Solution quality $\in [0, 1]$

Runtime in seconds

Simulations

Ideal

Noisy

Metrics

Solution quality $\in [0, 1]$

Runtime in seconds

Optimisation problems (#qubits $\in \{5, 6, \ldots, 19\}$)

Max-cut

Partition

Minimum vertex cover

Maximum independent set

$$(A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C)$$

Max-3SAT

Simulations

Ideal

Noisy

Metrics

Solution quality $\in [0, 1]$

Runtime in seconds

Optimisation problems (#qubits $\in \{5, 6, \ldots, 19\}$)

Max-cut

Partition

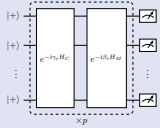
Minimum vertex cover

Maximum independent set

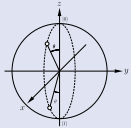
$$(A \vee \neg B \vee C) \wedge (\neg A \vee B \vee \neg C)$$

Max-3SAT

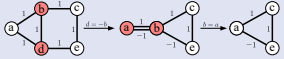
Algorithms (#layers $\in \{1, 2, \ldots, 7\}$)



QAOA

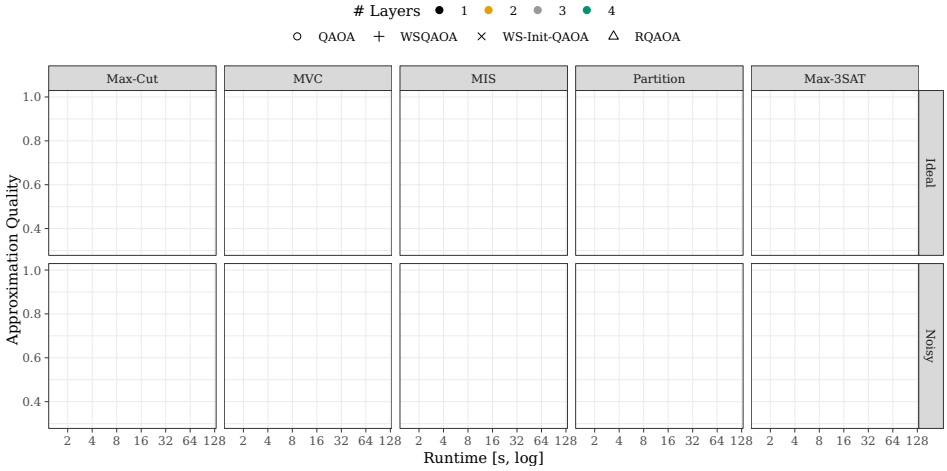


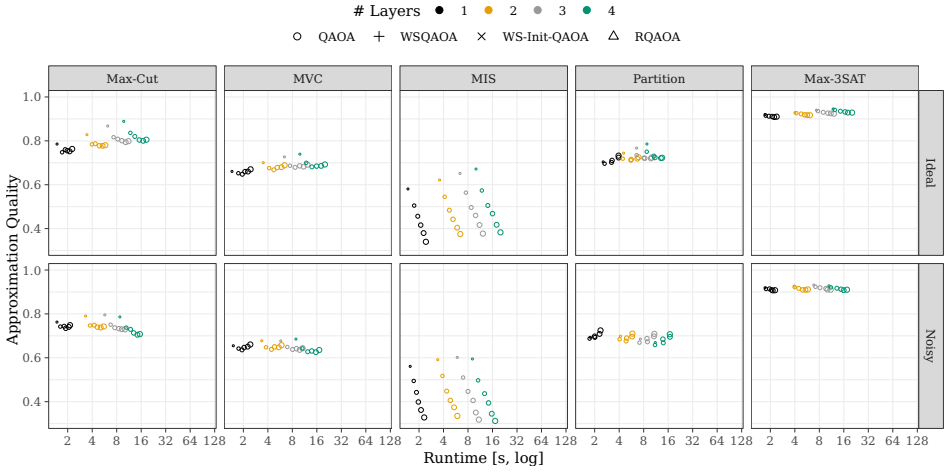
WSQAOA/WS-Init-QAOA

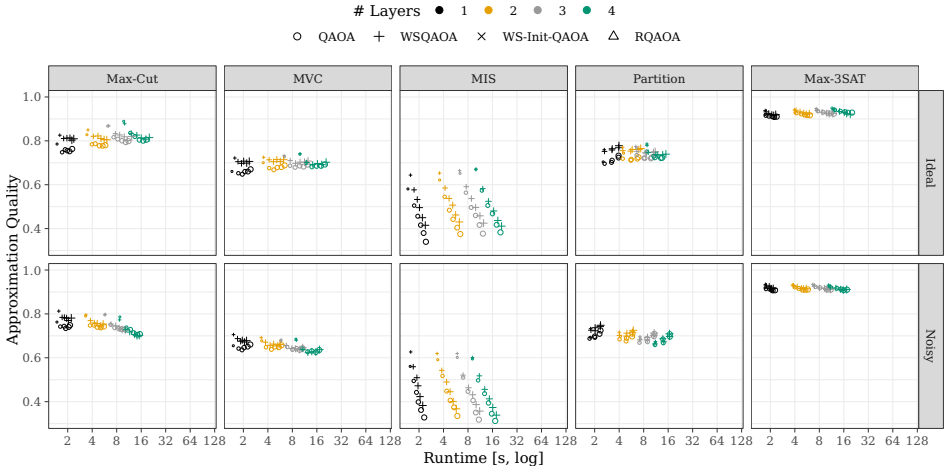


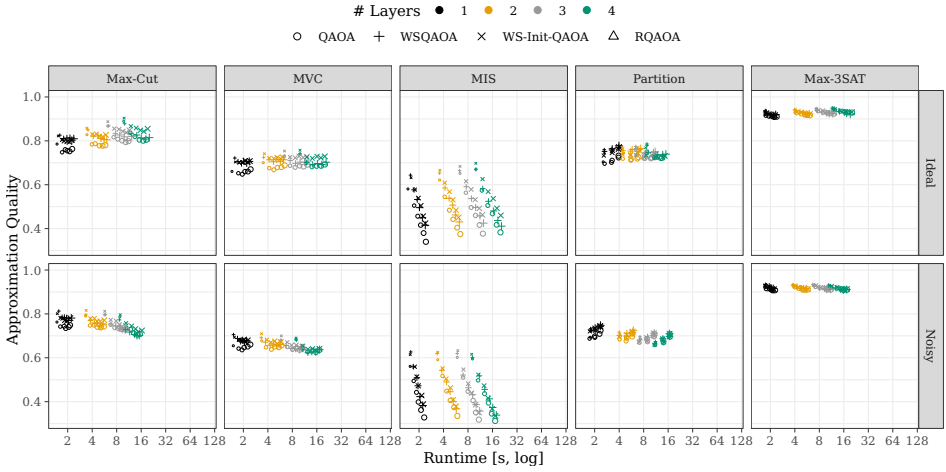
Recursive QAOA

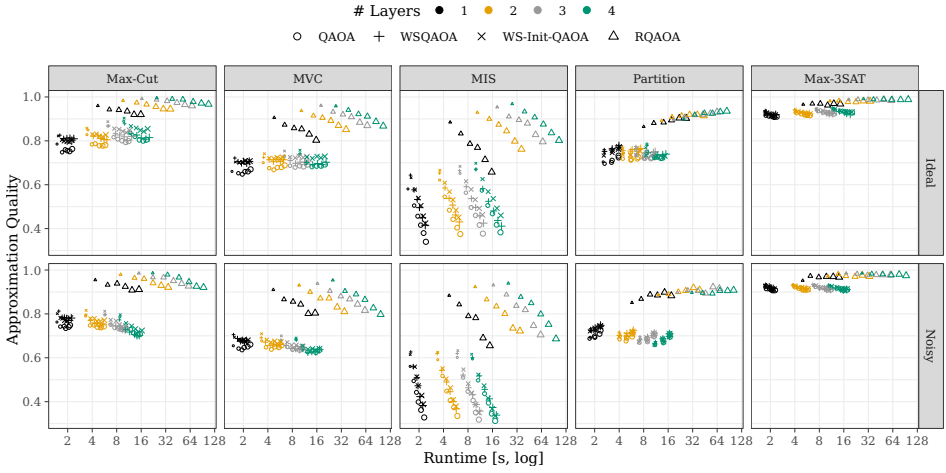
Experiment results




Layers 1 2 3 4
QAOA WSQAOA WS-Init-QAOA RQAOA
Max-Cut
MVC
MIS
Partition
Max-3SAT
Ideal
Noisy
Approximation Quality
1.0
0.8
0.6
0.4
2 4 8 16 32 64 128
Runtime [s, log]


Layers 1 2 3 4
QAOA WSQAOA WS-Init-QAOA RQAOA
Max-Cut
MVC
MIS
Partition
Max-3SAT
Ideal
Noisy
Approximation Quality
1.0
0.8
0.6
0.4
2 4 8 16 32 64 128
Runtime [s, log]


Layers 1 2 3 4
QAOA WSQAOA WS-Init-QAOA RQAOA
Max-Cut
MVC
MIS
Partition
Max-3SAT
Ideal
Noisy
Approximation Quality
1.0
0.8
0.6
0.4
2 4 8 16 32 64 128
Runtime [s, log]


Layers 1 2 3 4
QAOA WSQAOA WS-Init-QAOA RQAOA
Max-Cut
MVC
MIS
Partition
Max-3SAT
Ideal
Noisy
Approximation Quality
1.0
0.8
0.6
0.4
2 4 8 16 32 64 128
Runtime [s, log]

Predict algorithm *runtime* and *solution quality* accurately for *diverse quantum hardware*, using *few*, *small-scale* baseline instances.

For each problem, find bounds: $\mathrm{LB}(x) \leq f(x) \leq \mathrm{UB}(x)$

Normalize: $Y = \frac{f(x) - \mathrm{LB}(x)}{\mathrm{UB}(x) - \mathrm{LB}(x)}$

For each problem, find bounds: $\mathrm{LB}(x) \leq f(x) \leq \mathrm{UB}(x)$

Normalize: $Y = \frac{f(x) - \mathrm{LB}(x)}{\mathrm{UB}(x) - \mathrm{LB}(x)}$

Beta Regression

$$Y \approx \frac{1}{1 + \mathrm{e}^{-(\alpha + \beta n + \gamma d)}}$$

n: qubit count, d: layer count, b: baseline size

For each problem, find bounds: $\mathrm{LB}(x) \leq f(x) \leq \mathrm{UB}(x)$
Normalize: $Y = \frac{f(x) - \mathrm{LB}(x)}{\mathrm{UB}(x) - \mathrm{LB}(x)}$

Beta Regression

$$Y \approx \frac{1}{1 + \mathrm{e}^{-(\alpha + \beta n + \gamma d)}}$$

Power Law Decay

$$Y \approx \overline{Y}_d^{(b)} (1 + \alpha(n - b))^{\beta}$$

n: qubit count, d: layer count, b: baseline size

For each problem, find bounds: $\mathrm{LB}(x) \leq f(x) \leq \mathrm{UB}(x)$
Normalize: $Y = \frac{f(x)-\mathrm{LB}(x)}{\mathrm{UB}(x)-\mathrm{LB}(x)}$

Beta Regression

$$Y \approx \frac{1}{1 + \mathrm{e}^{-(\alpha+\beta n+\gamma d)}}$$

Power Law Decay

$$Y \approx \overline{Y}_d^{(b)}(1 + \alpha(n-b))^{\beta}$$

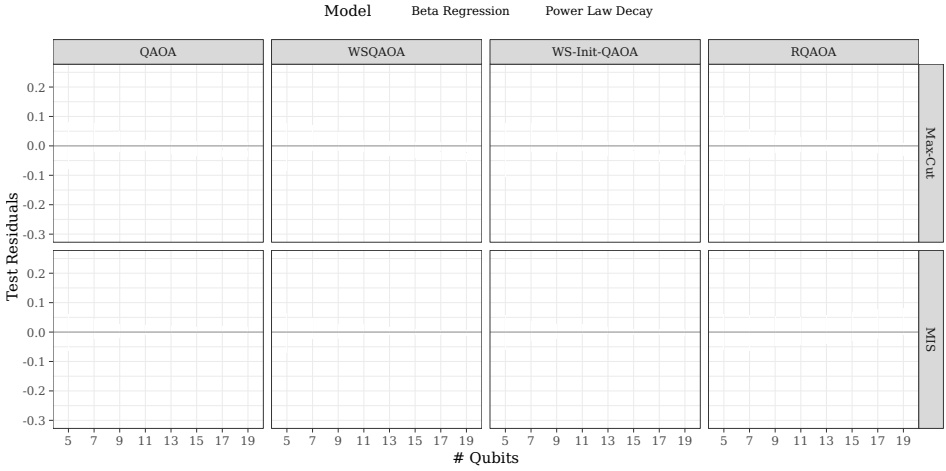
Noisy Quality Degradation

$$\begin{aligned} f_{\text{noisy}}(x) \approx \mathrm{LB}(x) \\ + (f_{\text{ideal}} - \mathrm{LB}(x)) \\ \cdot (1-l\beta)^{n d_{\text{C-X}}} \\ \cdot (1-l\gamma)^{n_{\text{C-X}}} \end{aligned}$$

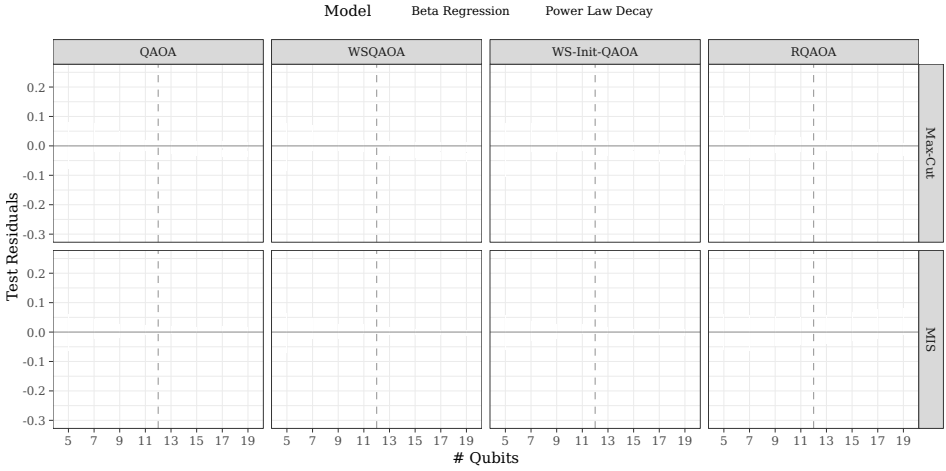
cf. Marshall et al. 2020, Xue et al. 2021

n: qubit count, d: layer count, b: baseline size

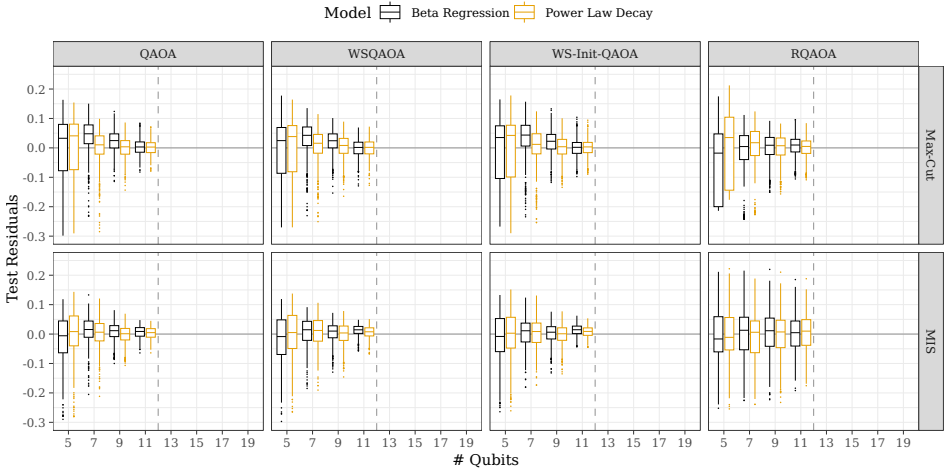
Model performance (ideal)



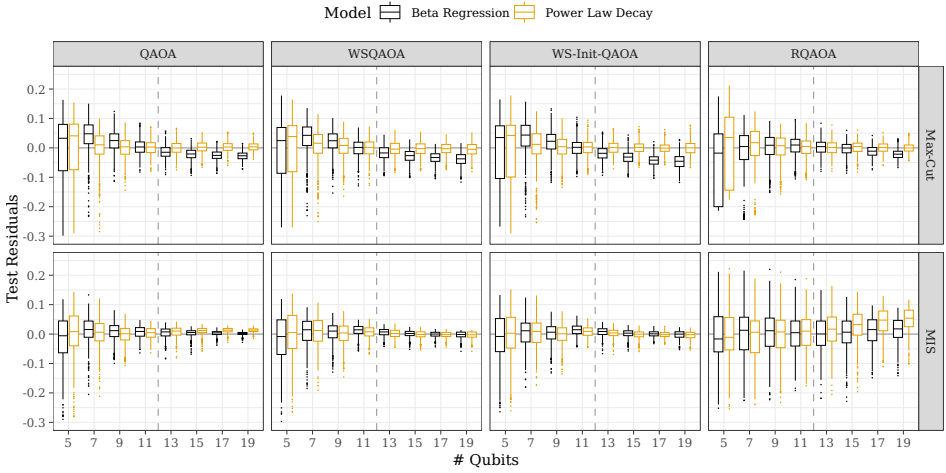
Model performance (ideal)



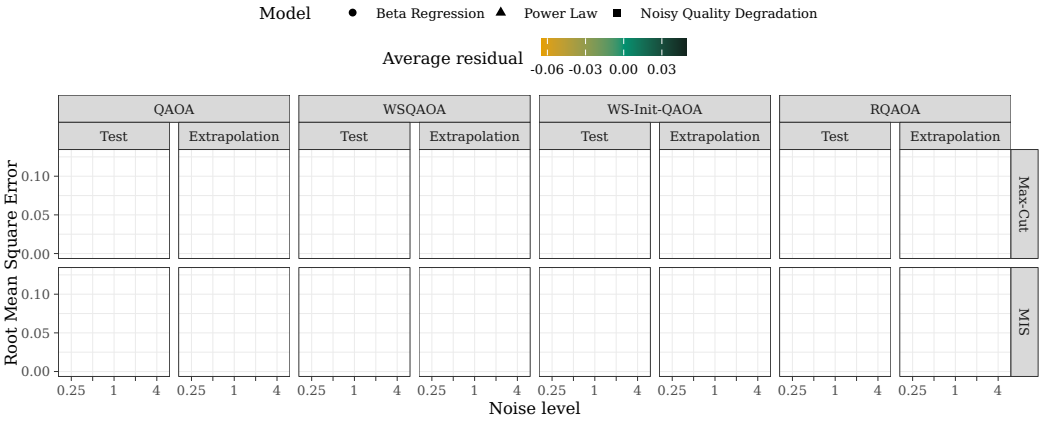
Model performance (ideal)

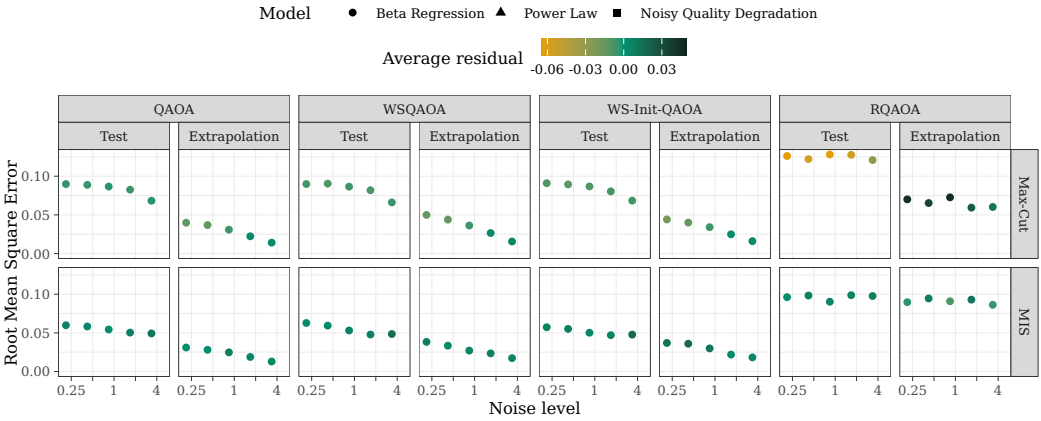


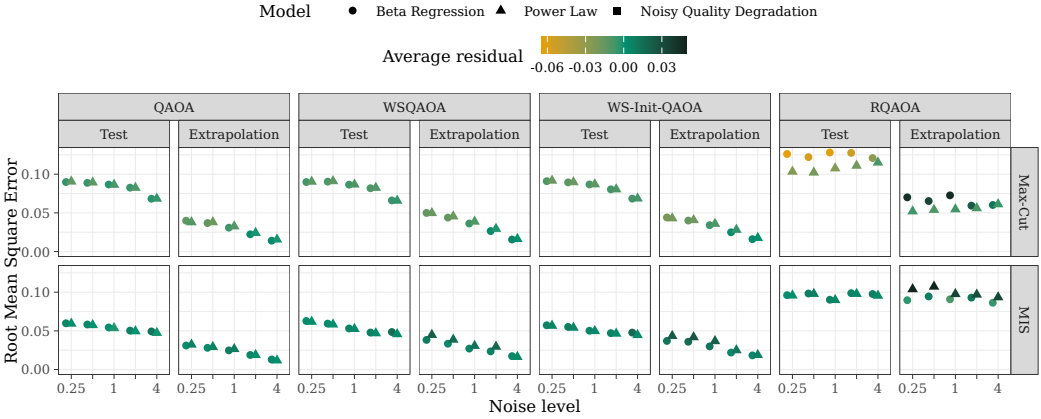
Model performance (ideal)

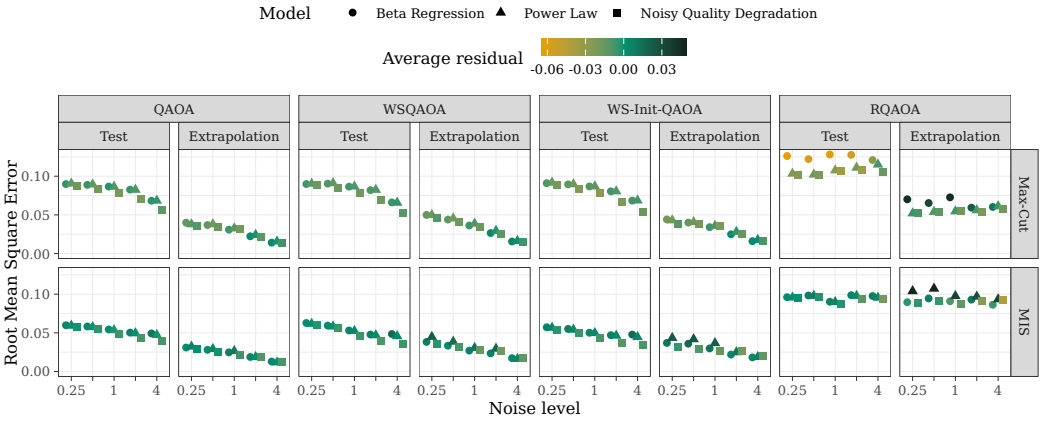


Model performance (noisy)

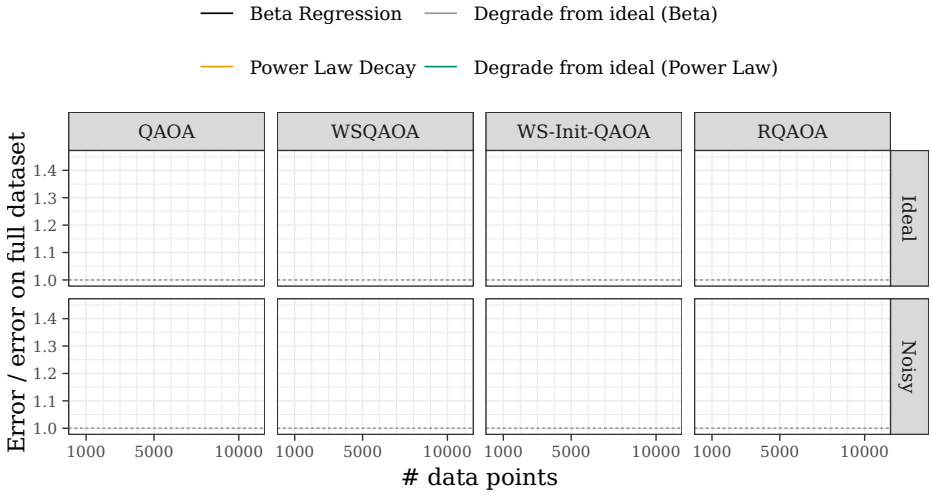



Model
Beta Regression
Power Law
Noisy Quality Degradation
Average residual
-0.06 -0.03 0.00 0.03
QAOA
WSQAOA
WS-Init-QAOA
RQAOA
Test
Extrapolation
Max-Cut
MIS
Root Mean Square Error
0.00
0.05
0.10
0.25
1
4
Noise level

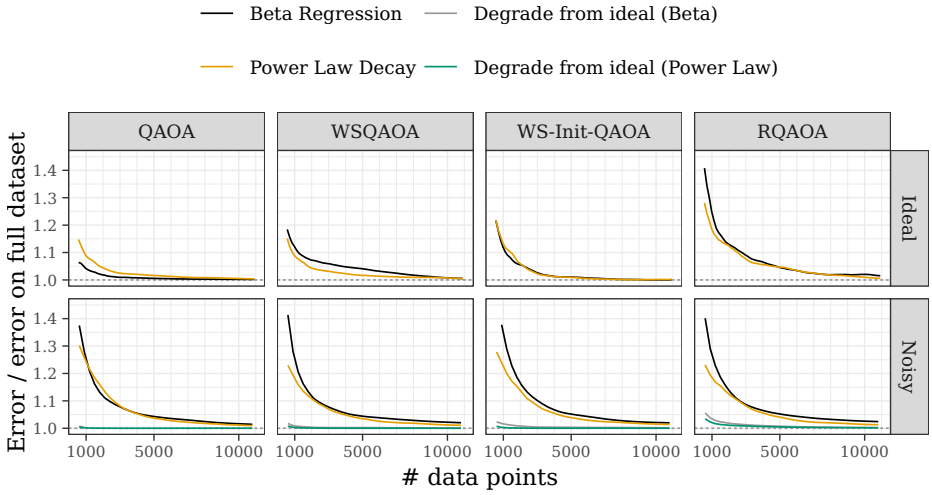

Model
Beta Regression
Power Law
Noisy Quality Degradation
Average residual
-0.06 -0.03 0.00 0.03
QAOA
WSQAOA
WS-Init-QAOA
RQAOA
Test
Extrapolation
Max-Cut
MIS
Root Mean Square Error
0.00
0.05
0.10
0.25
1
4
Noise level


Model
Beta Regression
Power Law
Noisy Quality Degradation
Average residual
-0.06 -0.03 0.00 0.03
QAOA
WSQAOA
WS-Init-QAOA
RQAOA
Test
Extrapolation
Max-Cut
MIS
Root Mean Square Error
0.00
0.05
0.10
0.25
1
4
Noise level

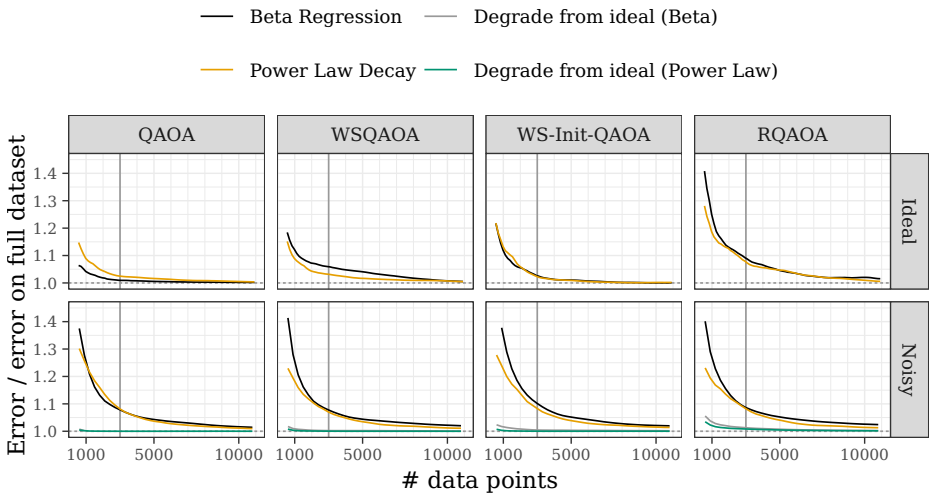
Effect of training dataset size



Effect of training dataset size



Effect of training dataset size



Predict algorithm *runtime* and *solution quality* accurately for *diverse quantum hardware*, using *few*, *small-scale* baseline instances.

Predict algorithm *runtime* and *solution quality* accurately for *diverse quantum hardware*, using *few*, *small-scale* baseline instances.

Software framework

`github.com/lfd/qce2025-design-automation`

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
```

Syntax proposal

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))
```

Syntax proposal

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))

with RELATIVE_SOLUTION_QUALITY >= 0.75:
```

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))

with RELATIVE_SOLUTION_QUALITY >= 0.75:
    with minimize(RUNTIME):
```

Syntax proposal

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))

with RELATIVE_SOLUTION_QUALITY >= 0.75:
    with minimize(RUNTIME):
        result = framework.quantum_solve(instance)
```

Syntax proposal

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))

with RELATIVE_SOLUTION_QUALITY >= 0.75:
    with minimize(RUNTIME):
        result = framework.quantum_solve(instance)

with maximize(SOLUTION_QUALITY), (
    (RUNTIME <= 10 * SECONDS) &
    (SOLUTION_QUALITY_PER_RUNTIME >= 100)
):
    result = framework.quantum_solve(instance)
```

Syntax proposal

```
from quantum_framework import *
import networkx as nx

framework = AlgorithmSelectionFramework()
framework.add_results(load_results())
instance = MaxCut(nx.erdos_renyi_graph(n = 30, p = 0.5))

with RELATIVE_SOLUTION_QUALITY >= 0.75:
    with minimize(RUNTIME):
        result = framework.quantum_solve(instance)

with maximize(SOLUTION_QUALITY), (
    (RUNTIME <= 10 * SECONDS) &
    (SOLUTION_QUALITY_PER_RUNTIME >= 100)
):
    result = framework.quantum_solve(instance)

with maximize(2/3 * SOLUTION_QUALITY - 1/3 * RUNTIME):
    result = framework.quantum_solve(instance)
```

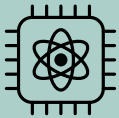
Framework architecture

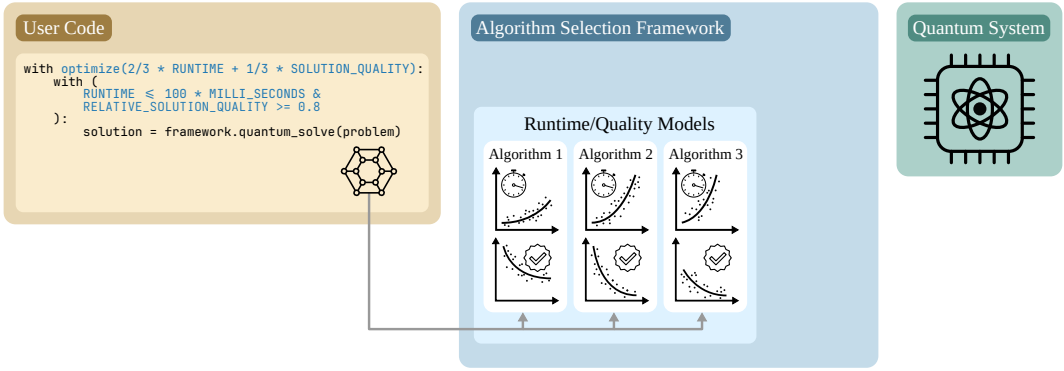
User Code

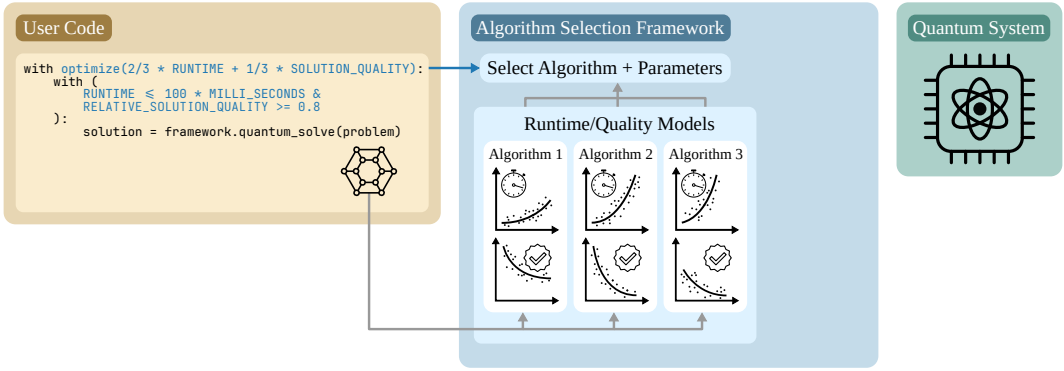
```
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
    with (
        RUNTIME ≤ 100 * MILLI_SECONDS &
        RELATIVE_SOLUTION_QUALITY >= 0.8
    ):
        solution = framework.quantum_solve(problem)
```

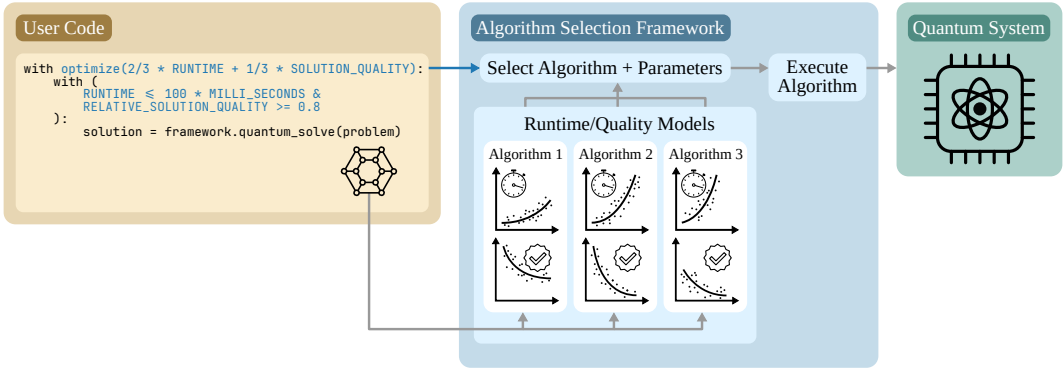
Algorithm Selection Framework

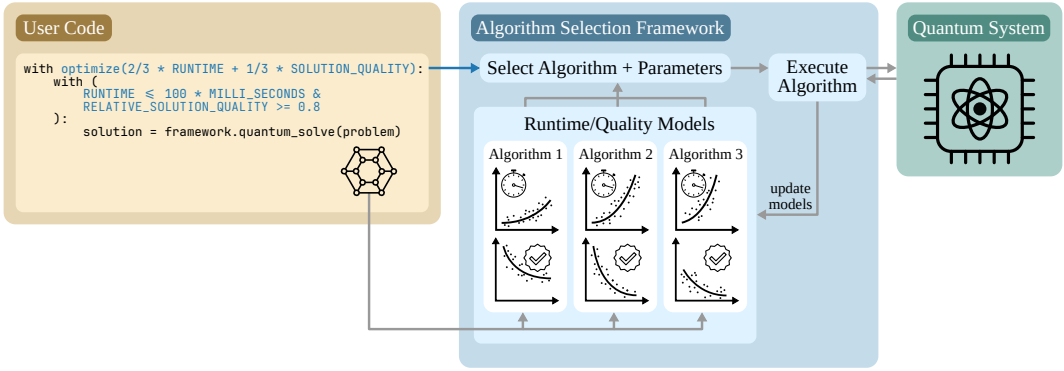
Quantum System

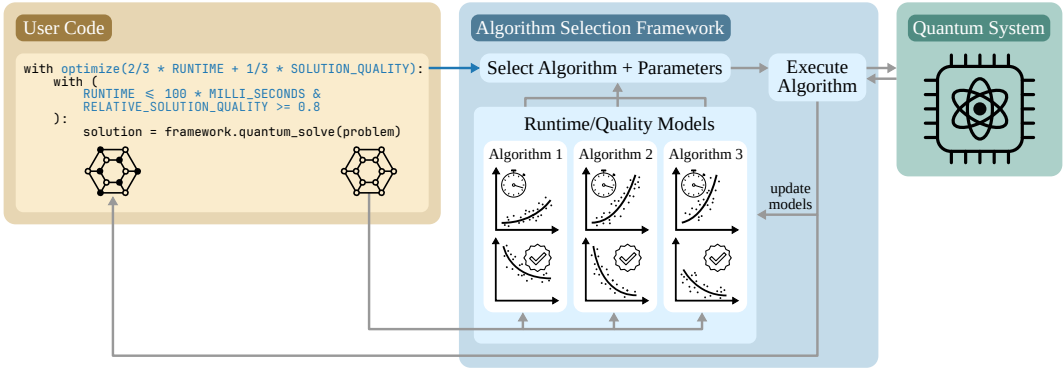



User Code
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
with (
RUNTIME ⩽ 100 * MILLI_SECONDS &
RELATIVE_SOLUTION_QUALITY >= 0.8
):
solution = framework.quantum_solve(problem)
Algorithm Selection Framework
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
Quantum System


User Code
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
with (
RUNTIME ⩽ 100 * MILLI_SECONDS &
RELATIVE_SOLUTION_QUALITY >= 0.8
):
solution = framework.quantum_solve(problem)
Algorithm Selection Framework
Select Algorithm + Parameters
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
Quantum System


User Code
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
with (
RUNTIME ⩽ 100 * MILLI_SECONDS &
RELATIVE_SOLUTION_QUALITY >= 0.8
):
solution = framework.quantum_solve(problem)
Algorithm Selection Framework
Select Algorithm + Parameters
Execute Algorithm
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
Quantum System


User Code
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
with (
RUNTIME ⩽ 100 * MILLI_SECONDS &
RELATIVE_SOLUTION_QUALITY >= 0.8
):
solution = framework.quantum_solve(problem)
Algorithm Selection Framework
Select Algorithm + Parameters
Execute Algorithm
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
update models
Quantum System


User Code
with optimize(2/3 * RUNTIME + 1/3 * SOLUTION_QUALITY):
with (
RUNTIME ≤ 100 * MILLI_SECONDS &
RELATIVE_SOLUTION_QUALITY >= 0.8
):
solution = framework.quantum_solve(problem)
Algorithm Selection Framework
Select Algorithm + Parameters
Execute Algorithm
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
update models
Quantum System

Use case: Hamiltonian simulation

Requirements

- Runtime
- Simulation error
- Quantum resources

Design decisions

- Approach
  (Trotter-Suzuki, Analog, Quantum Walk)
- Parameters
  (layer count, gate-based/analog hybrid)

Requirements

- Runtime
- Simulation error
- Quantum resources

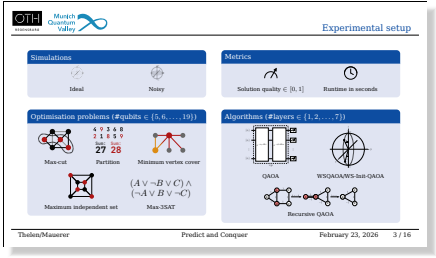
Design decisions

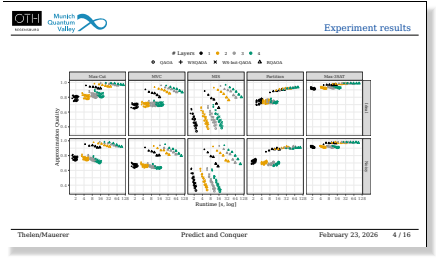
- Approach
  (Trotter-Suzuki, Analog, Quantum Walk)
- Parameters
  (layer count, gate-based/analog hybrid)

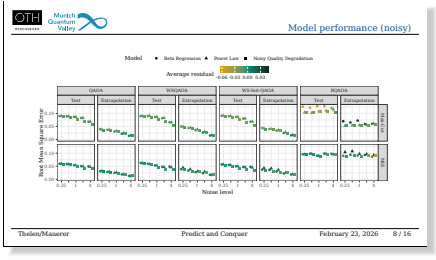
```
with maximize(INV_SIMULATION_ERROR_PER_RUNTIME), (
    SIMULATION_ERROR <= 0.001
):
    hamiltonian = Hamiltonian(...)
    result = framework.evolve_hamiltonian(
        hamiltonian,
        psi_0 = "0" * n, observable="Z" * n,
    )
```

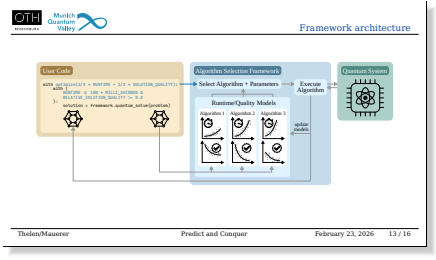
Future work

- Other problem classes
- Other influence factors
  - Number of measurements
  - Optimizer specification
  - ...
- Framework improvements


Experimental setup
Simulations
Ideal
Noisy
Metrics
Solution quality ∈ [0, 1]
Runtime in seconds
Optimisation problems (#qubits ∈ {5, 6, . . . , 19})
Max-cut
Partition
Minimum vertex cover
Maximum independent set
Max-3SAT
Algorithms (#layers ∈ {1, 2, . . . , 7})
QAOA
WSQAOA/WS-Init-QAOA
Recursive QAOA
Thelen/Mauerer
Predict and Conquer
February 23, 2026
3 / 16


Experiment results
Layers
QAOA
WSQAOA
WS-Init-QAOA
RQAOA
Max-Cut
MVC
MIS
Partition
Max-3SAT
Approximation Quality
Runtime [s, log]
Thelen/Mauerer
Predict and Conquer
February 23, 2026
4 / 16


Model performance (noisy)
Model
Beta Regression
Power Law
Noisy Quality Degradation
Average residual
QAOA
WSQAOA
WS-Init-QAOA
RQAOA
Test
Extrapolation
Root Mean Square Error
Noise level
Thelen/Mauerer
Predict and Conquer
February 23, 2026
8 / 16


Framework architecture
User Code
Algorithm Selection Framework
Select Algorithm + Parameters
Execute Algorithm
Runtime/Quality Models
Algorithm 1
Algorithm 2
Algorithm 3
update models
Quantum System
Thelen/Mauerer
Predict and Conquer
February 23, 2026
13 / 16

Backup Slides

Runtime model

$$T(x) \approx \alpha d_{\text{C-}X}^{\beta}$$

Runtime model

$$T(x) \approx \alpha d_{\text{C-}X}^{\beta}$$