Supplementary Information

Generative quantum combinatorial optimization by means of a novel conditional generative quantum eigensolver

Shunya Minami[1*], Kouhei Nakaji[2,3], Yohichi Suzuki[1], Alán Aspuru-Guzik[3,4], and Tadashi Kadowaki[1,5]

[1]Global R&D Center for Business by Quantum-AI Technology, National Institute of Advanced Industrial Science and Technology, Ibaraki, Japan
[2]NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, CA 95051, USA
[3]Chemical Physics Theory Group, Department of Chemistry, University of Toronto, Toronto, Ontario, Canada
[4]Vector Institute for Artificial Intelligence, Toronto, Ontario, Canada
[5]DENSO CORPORATION, Tokyo, Japan
[*]*Corresponding author: s-minami@aist.go.jp*

GQCO model architecture and hyperparameter settings

Fig. S1 illustrates the model structure, and Table S1 lists the hyperparameter settings used in this work. As mentioned in the main text, the entire model consists of 256 million parameters, with 127 million allocated to the encoder and 129 million to the decoder. Note that this total number includes parameters for all expert modules up to size 20. The number of parameters utilized for a specific problem size is approximately 24 million, with 11 million in the encoder and 13 million in the decoder.

During DPO training, parameters are updated to increase the probability of generating the most preferred circuit. Sampling a large number of circuits at once increases the probability of obtaining circuits near the ground state, thereby enhancing training efficiency. In this study, for each number of qubits n, the value of M was determined to maximize GPU memory usage. These settings are also detailed in Table S1.

Training history

A multi-GPU strategy was used to train the model in a distributed environment. Hamiltonian coefficients were generated independently on each GPU, losses were computed locally, and gradients were aggregated across all GPUs to update the model parameters. The number of GPUs varied based on available computational resources, ranging from 64 to 256 during training. The training process took approximately 20 days.

Fig. S2a illustrates the total computational cost for each training size stage, calculated by multiplying the training time per GPU by the number of GPUs used. This metric represents the process time required to train on a single GPU. Fig. S2b shows the product of the number of training epochs and the number of GPUs, which corresponds to the total number of Hamiltonian coefficients generated during training, that is, the number of combinatorial optimization problems processed during training. Both metrics increase exponentially as the maximum problem size grows. As the number of qubits increases, a greater number of training instances are required, and quantum circuit simulations take longer.

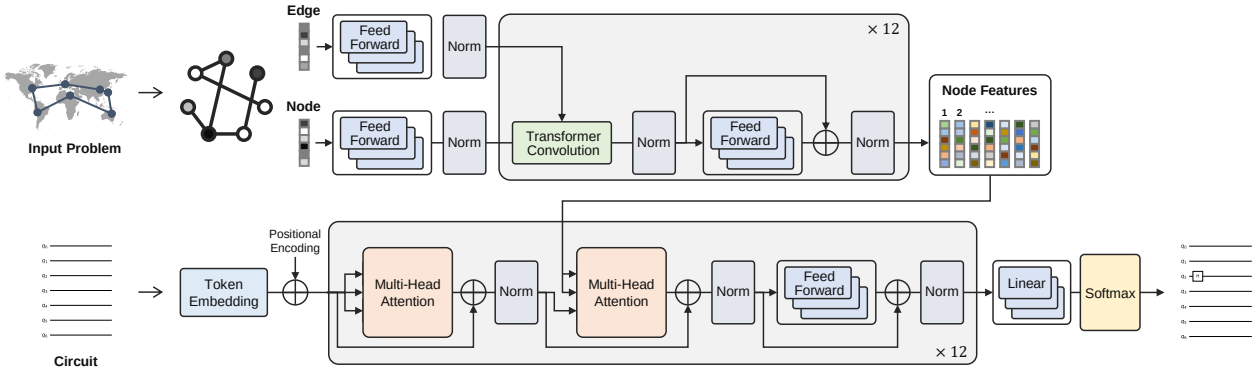


Figure S1: Model archtecture. As shown in the Results section in the main text, the input combinatorial optimization problem is converted into a graph representation, and the features are engineered. The module containing the Graph Transformer layer is iterated 12 times, and the resulting node features, arranged in node order, serve as the encoding representation. The decoder resembles the structure used in GPT models: `FeedForward` refers to two linear layers with an activation function between them, and `Norm` refers to the layer normalization layer. The four `FeedForward` modules and one linear layer module are organized in a mixture-of-experts (MoE) structure.

Table S1: Hyperparameter settings for model architecture and training. n represents the problem size, i.e., the number of qubits used in the circuit to be generated. For the hyperparameters that are not listed, the default settings of PyTorch, PyTorch Geometric, and PyTorch Lightning were used.

	Hyperparameter		Value
Base settings	Activation function		GELU
	Dropout rate		0.0
	ϵ for layer normalization		1×10^{-5}
Encoder	Depth		12
	Number of attention heads		8
	Dimension of the intermediate representations		256
	Dimension of the intermediate feed-forward layer		1024
Decoder	Depth		12
	Number of attention heads		8
	Dimension of the intermediate representations		256
	Dimension of the intermediate feed-forward layer		1024
Training settings	Optimizer		Adam
	Learning rate		1×10^{-4}
	(β_1, β_2) for Adam optimizer		$(0.9, 0.95)$
	β for DPO		0.1
	Evaluation frequency		500
Generation settings	Maximum sequence length		$2n$
	Vocabulary size		$4n^2 + 15 + 1$
	Number of samplings during training M	for $n = 3$	1024
		for $n = 4$	1024
		for $n = 5$	512
		for $n = 6$	384
		for $n = 7$	256
		for $n = 8$	192
		for $n = 9$	128
		for $n = 10$	96

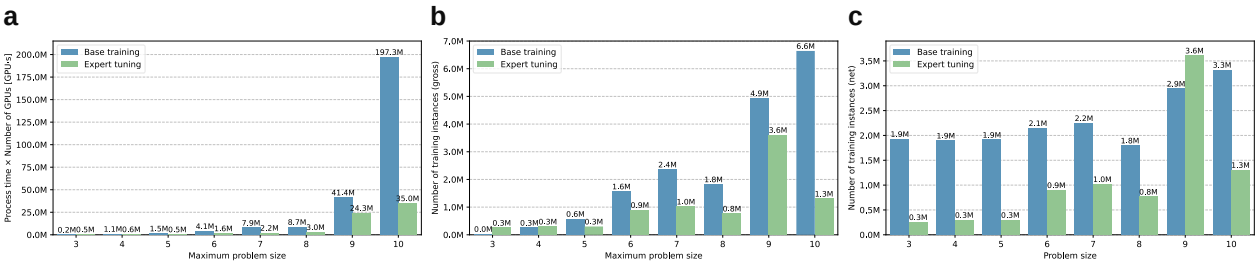


Figure S2: Computational costs for training across problem sizes. Blue bars represent the costs for base training phase where all the weights of the model are optimized, while green bars indicate the costs for expert tuning phase where each expert in the MoE layer is fine-tuned after the base training. (a) Total computational time for each curriculum learning stage. The value is calculated by multiplying the process time for distributed training by the number of GPUs utilized during the corresponding stage. (b) Gross amount of training data across problem size. The total number of training data points is determined by multiplying the number of GPUs by the number of training epochs. The value at each maximum problem size represents the total data generated for that stage, including all problem sizes up to the corresponding size. (c) Net amount of training data across problem size. This value is derived by adjusting the values of (b) to reflect the net number of data points for each problem size, based on the selection probability Eq. (4) of each size.

It should be noted that, at each problem size stage, problems up to the maximum size are generated based on the probability Eq. (4) in the main text. As a result, the number of training instances in Fig. S2b includes various problem sizes. Correcting this, we show the net number of training instances for each specific problem size in Fig. S2c. This reveals that the number of training examples increases more slowly with problem size, suggesting that curriculum learning and MoE structures may enhance learning efficiency. However, the probability of generating each problem size based on Eq. (4) is not optimized and may produce an excessive number of instances with small problem size. In other words, fewer instances than those shown in Fig. S2c might be sufficient for learning smaller problem sizes, and the observed scaling in Fig. S2c may be overly gradual.

Pseudocode for the training procedure

To clarify the training process further, we present the pseudocode in Algorithm S1.

Algorithm S1 GQCO Training Procedure

Require: n_{sample}: Number of circuit configurations sampled per iteration
`RandomSize`$(\cdot)$: Function sampling circuit size based on Eq. (4) in the main text
`RandomAdjacencyMatrix`$(\cdot)$: Function generating random adjacency matrix for given size
`Hamiltonian`$(\cdot)$: Function constructing Hamiltonian from adjacency matrix
`Model`$(\cdot;\theta)$: Parameterized model generating tokens
`Detokenize`$(\cdot)$: Function converting tokens into quantum circuit unitary operators
`ComputeEnergy`$(\cdot,\cdot)$: Function calculating energies using given Hamiltonian and circuit
`Loss`$(\cdot)$: Function calculating loss according to Eq. (3) in the main text
`Update`$(\cdot)$: Parameter update function based on optimization algorithm

Initialize $max\_size = 3$, θ_t
while not converged **do**
  $size \leftarrow$ `RandomSize`(max_size)
  $A \leftarrow$ `RandomAdjacencyMatrix`$(size)$
  $H \leftarrow$ `Hamiltonian`(A)
  $\{t_1, t_2, \ldots, t_{n_{\text{sample}}}\} \leftarrow$ `Model`$([A] \times n_{\text{sample}}; \theta_t)$
  $\{U_1, U_2, \ldots, U_{n_{\text{sample}}}\} \leftarrow$ `Detokenize`$(\{t_1, t_2, \ldots, t_{n_{\text{sample}}}\})$
  $\{E_1, E_2, \ldots, E_{n_{\text{sample}}}\} \leftarrow$ `ComputeEnergy`$(\{U_1, U_2, \ldots, U_{n_{\text{sample}}}\}, H)$
  $w_{\text{best}} \leftarrow \arg\min_i \{E_1, E_2, \ldots, E_{n_{\text{sample}}}\}$
  $\mathcal{L} \leftarrow$ `Loss`(w_{best})
  $\theta_{t+1} \leftarrow$ `Update`(θ_t)
  if convergence criterion met at current max_size **then**
    $max\_size \leftarrow max\_size + 1$
  end if
end while

Additional figures

Enlarged figures for the real device execution

The enlarged versions of Fig. 4c, displaying the results obtained from the real quantum device, are shown in Fig. S3 for GQCO and Fig. S4 for QAOA.

Examples of generated circuits

In Fig. S5, we present examples of GQCO-generated circuits for each number of qubits.

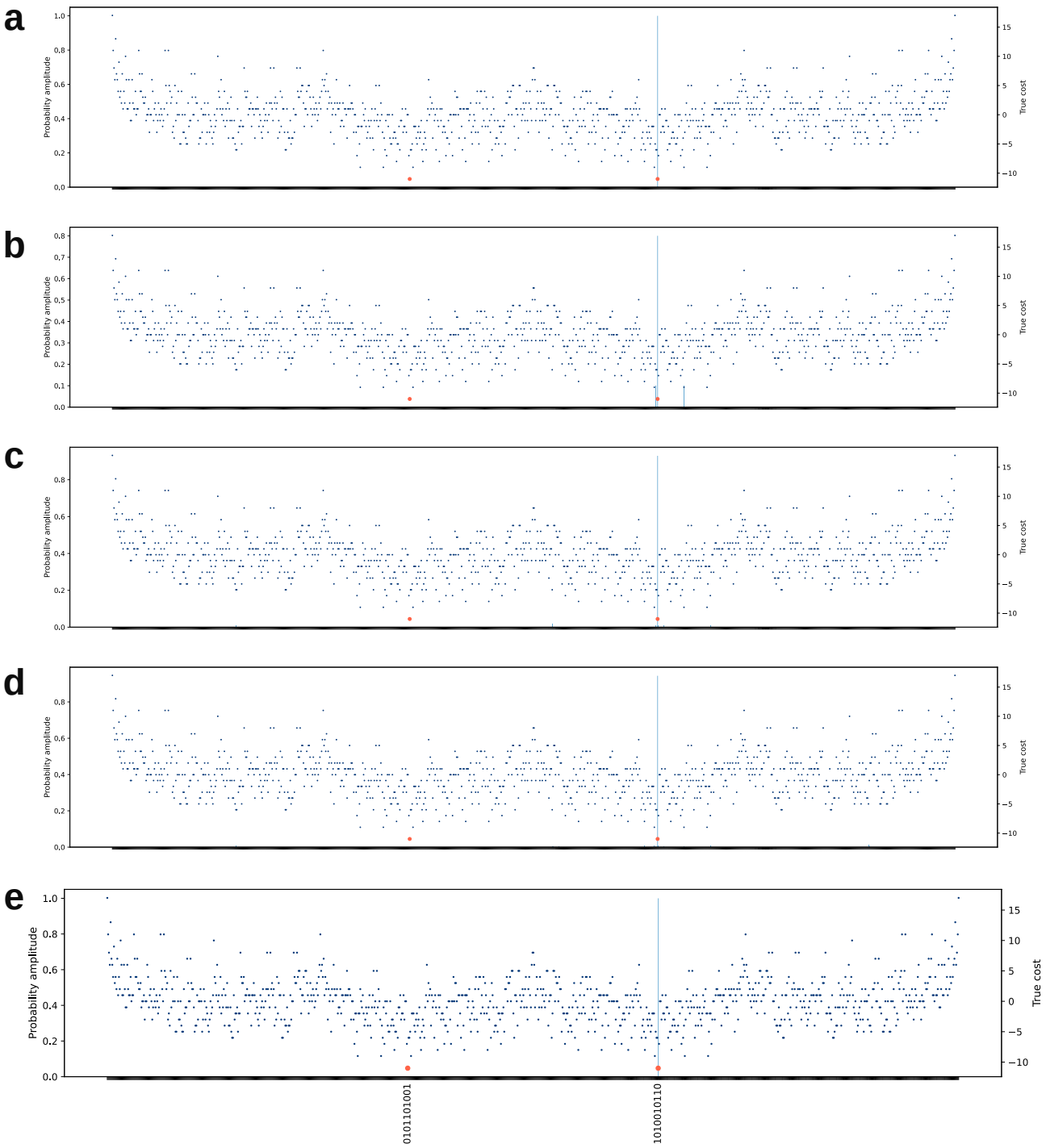


Figure S3: Sampling results on the real quantum device for the GQCO-generated circuit. (a) 1 shot, (b) 10 shots, (c) 100 shots, (d) 1000 shots, and (e) simulated state vector are displayed. The meanings of the plots and histograms are the same as that of Fig. 4c.

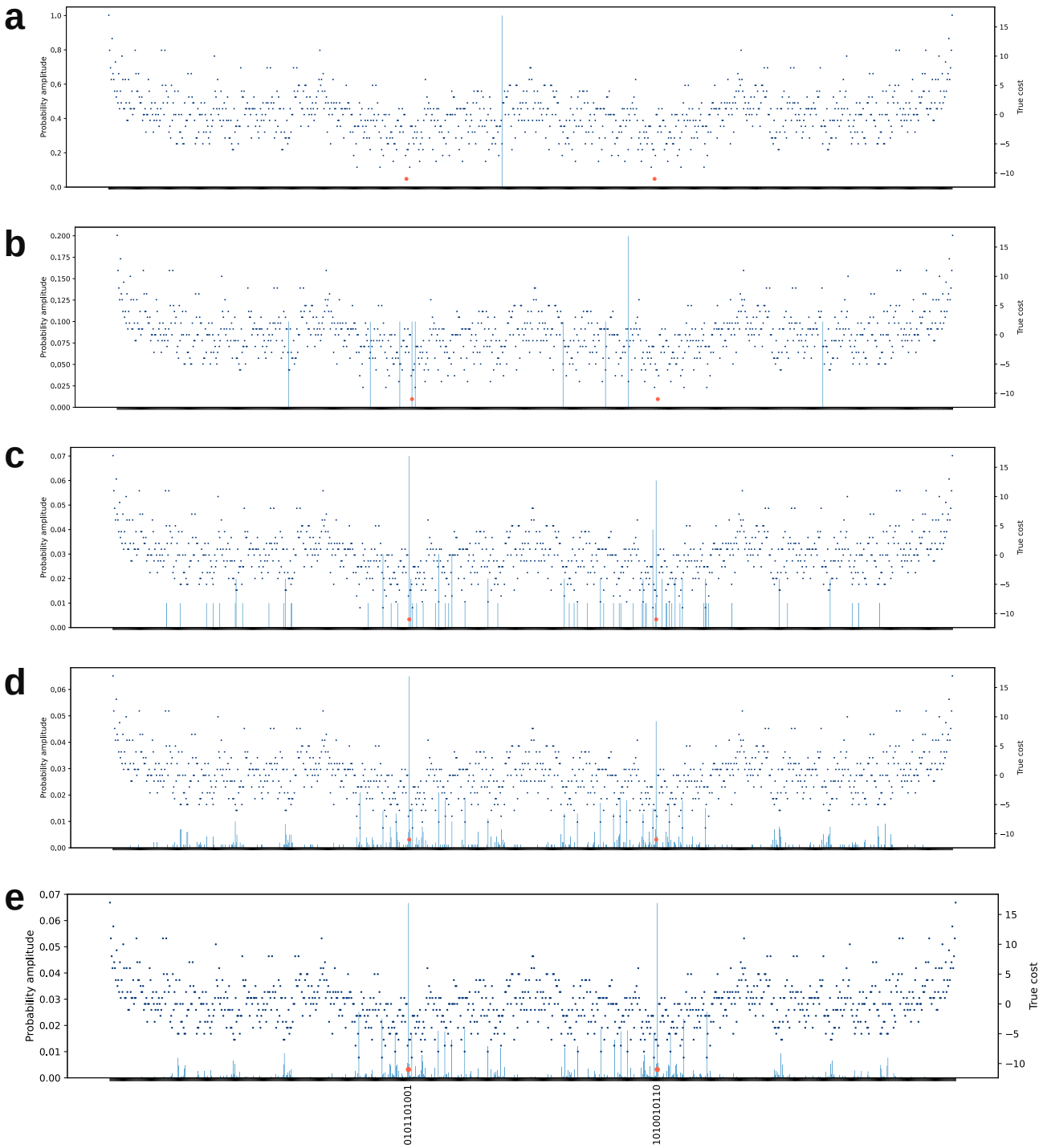
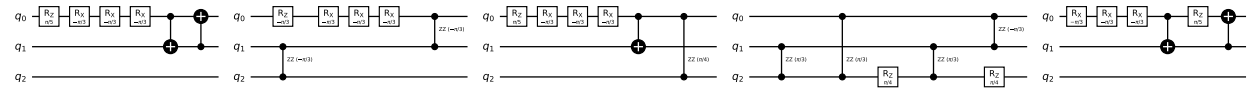
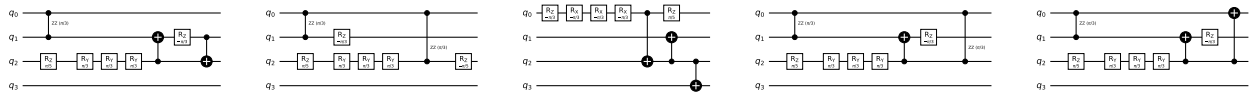


Figure S4: Sampling results on the real quantum device for the 2-layer QAOA circuit. (a) 1 shot, (b) 10 shots, (c) 100 shots, (d) 1000 shots, and (e) simulated state vector are displayed. The meanings of the plots and histograms are the same as that of Fig. 4c.

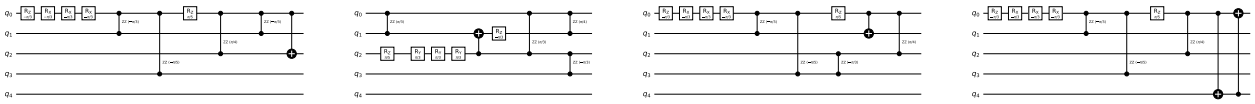
Size3



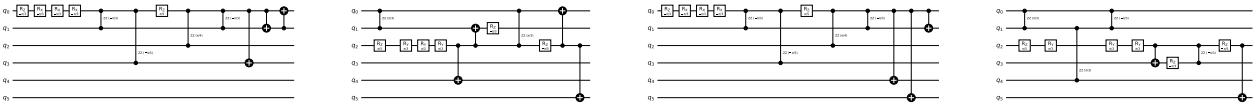
Size4



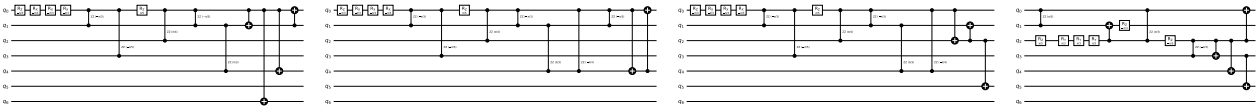
Size5



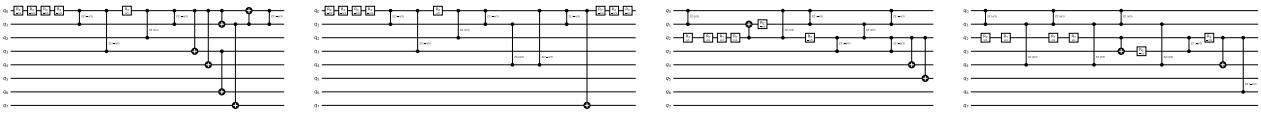
Size6



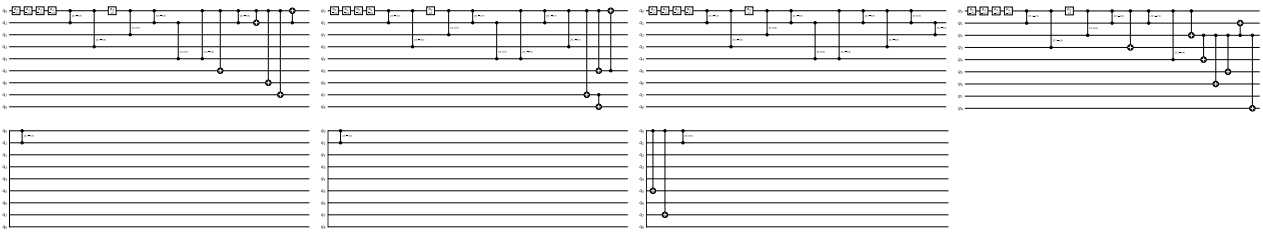
Size7



Size8



Size9



Size10

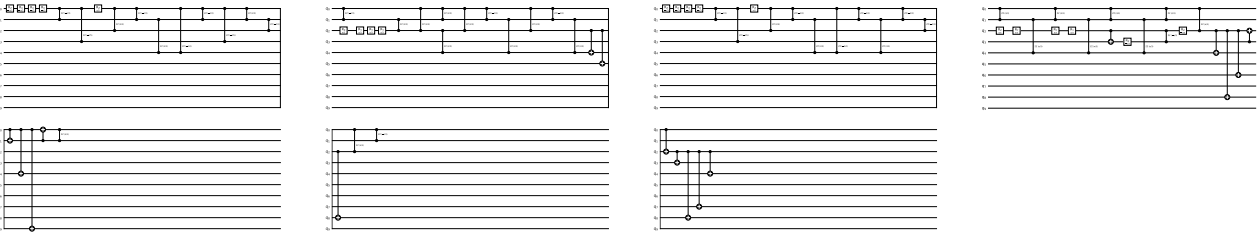


Figure S5: Examples of GQCO-generated quantum circuits.