In [1]:
```
import qiskit

qiskit.__version__
```

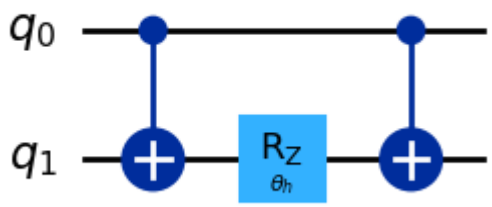
Out[1]: '1.4.2'

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
import rustworkx as rx

from qiskit import QuantumCircuit, transpile
from qiskit.circuit import Parameter
from qiskit.circuit.library import YGate
from qiskit.quantum_info import SparsePauliOp
from qiskit_ibm_runtime import QiskitRuntimeService, fake_provider, EstimatorV2 as Estimator
from qiskit_aer import AerSimulator
```

In [3]:
```
service = QiskitRuntimeService()
```

In [5]:
```
from qiskit.circuit.library import RZZGate
θ_h = Parameter("$\\theta_h$")
qc1= QuantumCircuit(2)
qc1.append(RZZGate(θ_h),[0, 1])
qc1.decompose(reps=1).draw("mpl")
```

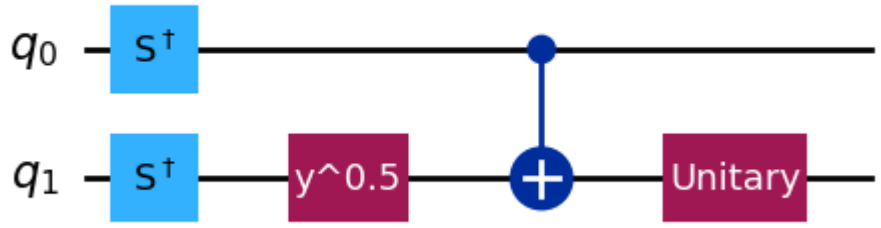
Out[5]:



In [6]:

```
qc2 = QuantumCircuit(2)

qc2.sdg([0, 1])
qc2.append(YGate().power(1/2), [1])
qc2.cx(0, 1)
qc2.append(YGate().power(1/2).adjoint(), [1])

qc2.draw("mpl")
```

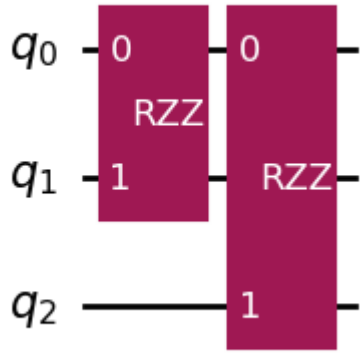
Out[6]:



In [7]:

```
rzz = qc2.to_gate(label="RZZ")
```

In [8]:
```
qc3 = QuantumCircuit(3)
qc3.append(rzz, [0,1])
qc3.append(rzz, [0,2])
display(qc3.draw("mpl"))
#display(qc.decompose(reps=1).draw("mpl"))
```



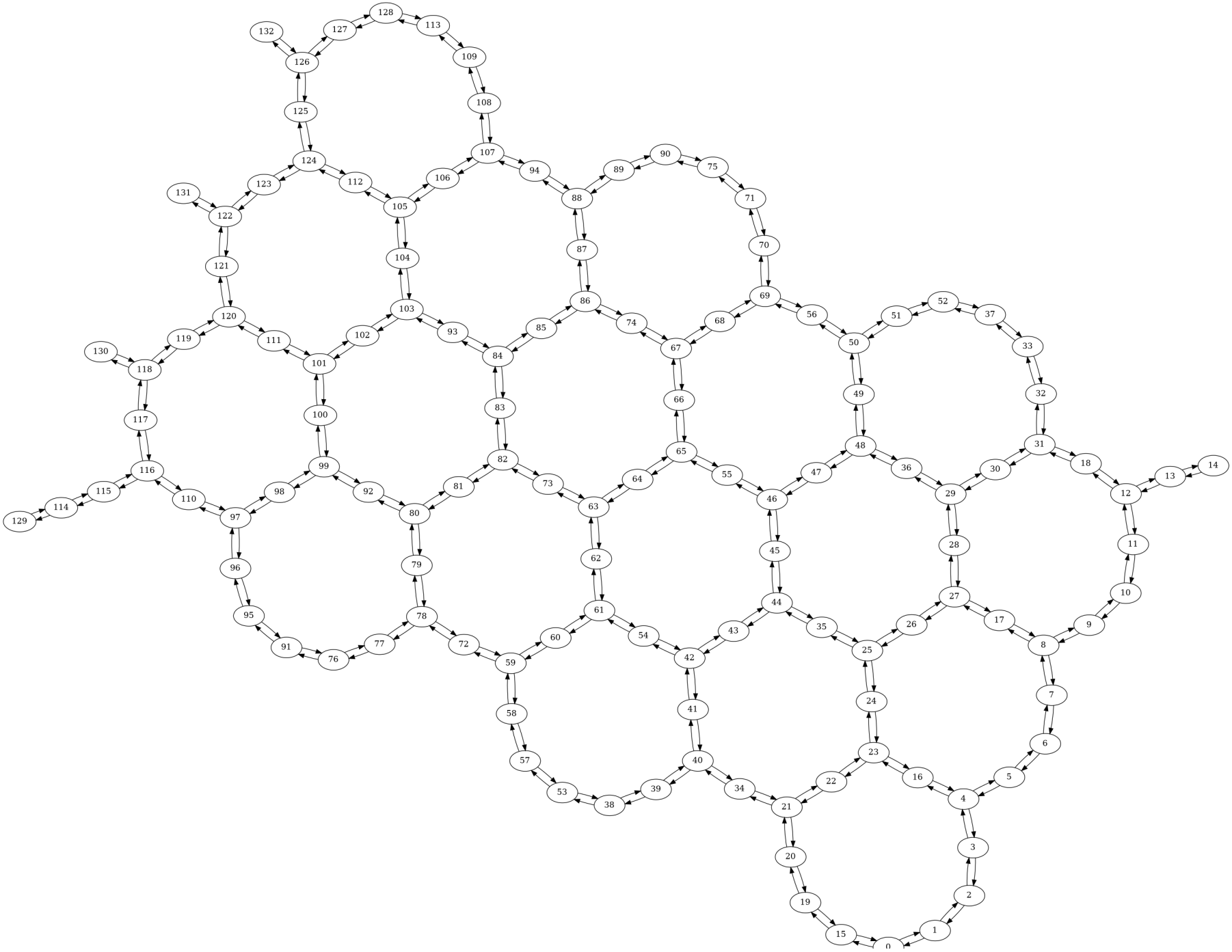
In [9]:
```
from qiskit.quantum_info import Operator

op1 = Operator(qc1.assign_parameters([-np.pi/2]))
op2 = Operator(qc2)

op1.equiv(op2)
```

Out[9]: True

In [11]:

```
backend = service.backend("ibm_torino")
backend.coupling_map.draw()
```

Out[11]:


128
113
132
127
109
126
108
125
107
124
94
90
89
75
106
112
123
131
88
71
105
122
87
70
104
121
86
69
103
120
74
56
52
85
68
51
37
93
111
119
102
67
50
33
130
84
101
118
66
49
32
83
100
117
65
48
31
82
99
116
55
47
36
18
14
73
64
30
13
115
81
98
46
29
12
92
63
110
114
80
45
28
11
97
62
129
79
44
27
10
96
61
35
26
17
9
95
78
54
43
25
8
91
77
72
60
42
24
7
76
59
41
23
6
58
40
22
16
5
57
53
39
34
4
38
21
3
20
2
19
15
1
0

In [12]:
```python
def color_coupling_map(backend):
    graph = backend.coupling_map.graph
    undirected_graph = graph.to_undirected(multigraph=False)
    edge_color_map = rx.graph_bipartite_edge_color(undirected_graph)
    if (edge_color_map is None):
        edge_color_map = rx.graph_greedy_edge_color(undirected_graph)
    # build a map from color to a list of edges
    edge_index_map = undirected_graph.edge_index_map()
    color_edges_map = {color: [] for color in edge_color_map.values()}
    for edge_index, color in edge_color_map.items():
        color_edges_map[color].append((edge_index_map[edge_index][0], edge_index_map[edge_index][1]))
    return edge_color_map, color_edges_map
```

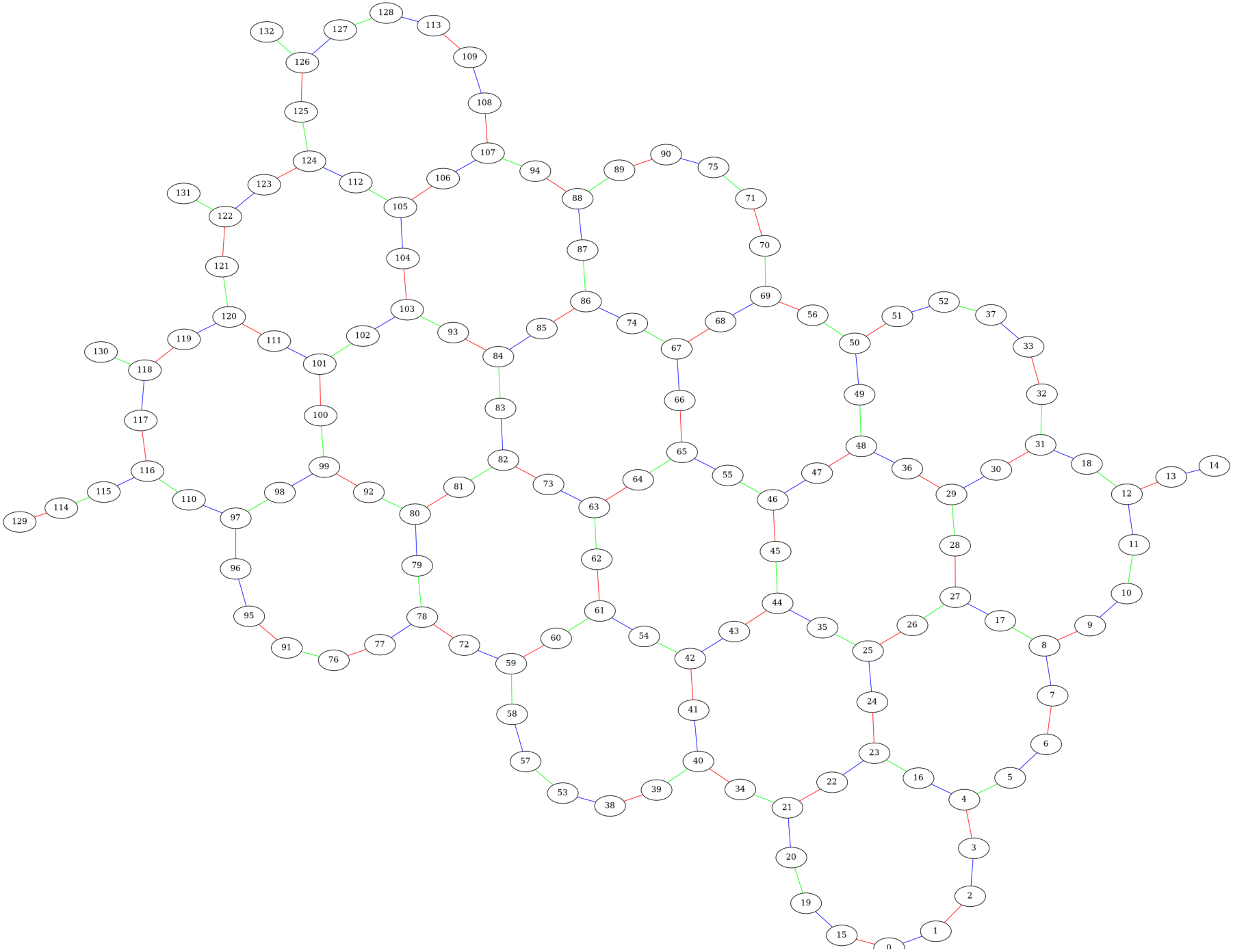
In [13]:
```python
edge_color_map, color_edges_map = color_coupling_map(backend)
print(f"{backend.name}, {backend.num_qubits}-qubit device, {len(color_edges_map.keys())} colors assigne
```

```
ibm_torino, 133-qubit device, 3 colors assigned.
```

In [14]:

```
color_str_map = {0: "green", 1: "red", 2: "blue"}

undirected_graph = backend.coupling_map.graph.to_undirected(multigraph=False)
for i in undirected_graph.edge_indices():
    undirected_graph.get_edge_data_by_index(i)["color"] = color_str_map[edge_color_map[i]]

rx.visualization.graphviz_draw(undirected_graph, method="neato", edge_attr_fn=lambda edge: {"color": ed
```

Out[14]:


128
113
132
127
109
126
108
125
107
124
90
89
75
94
106
123
112
88
71
131
105
122
87
70
104
121
86
69
103
52
120
74
68
56
51
37
102
85
119
111
93
67
50
33
130
101
84
66
49
32
118
83
100
48
31
117
65
82
99
18
116
64
55
47
36
30
14
115
73
81
98
92
63
46
29
12
13
110
80
45
28
11
114
97
62
27
10
129
79
44
61
26
17
9
96
78
43
35
25
8
95
60
54
42
24
7
91
77
72
41
6
76
59
23
16
5
58
40
22
4
57
39
34
21
3
53
38
20
2
19
1
15
0

In [15]:

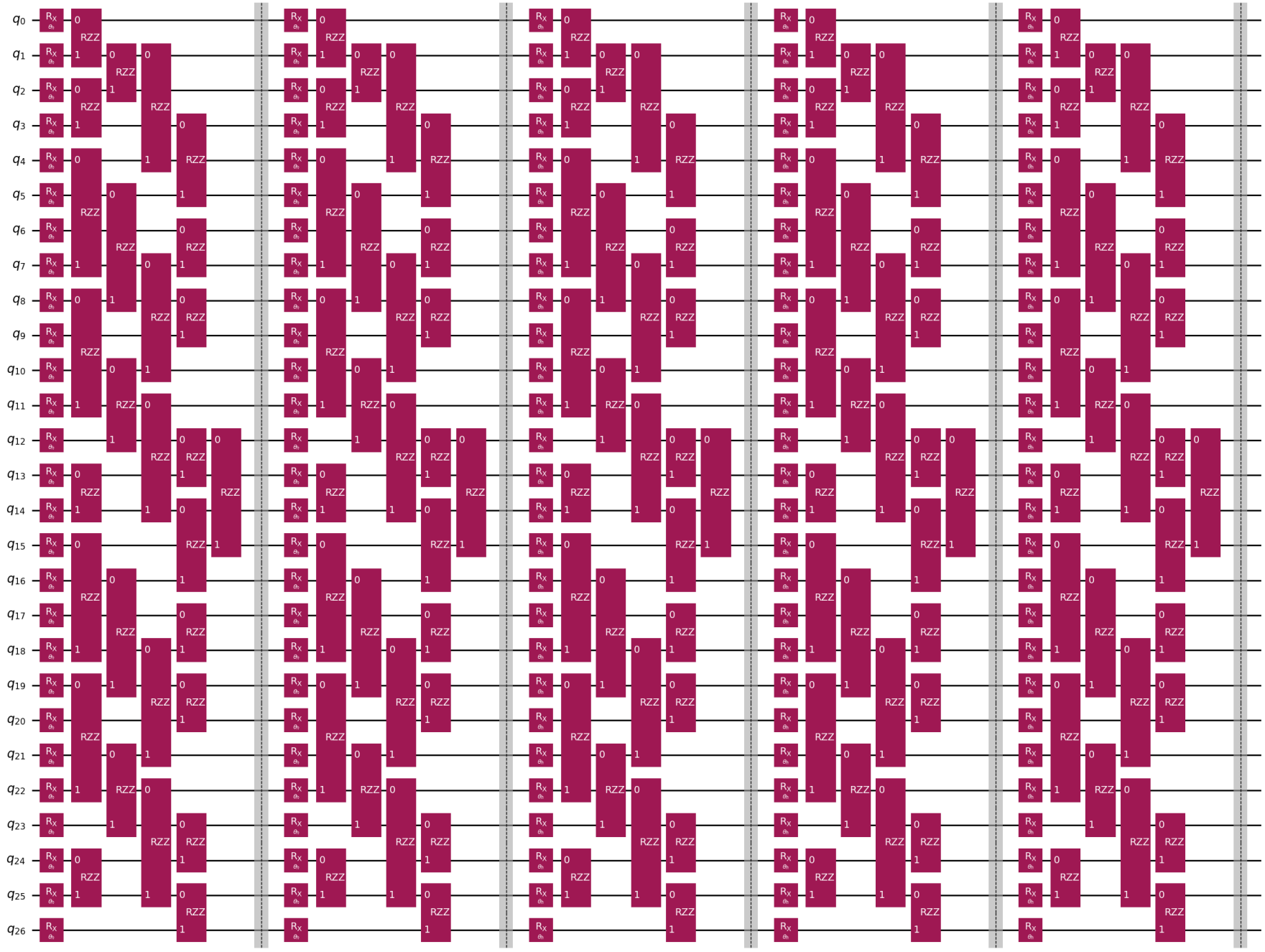
```python
def get_utility_circuit(backend, num_steps: int, barrier: bool = False):
    num_qubits = backend.num_qubits
    _, color_edges_map = color_coupling_map(backend)
    θ_h = Parameter("$\\theta_h$")
    qc = QuantumCircuit(num_qubits)

    for i in range(num_steps):
        qc.rx(θ_h, range(num_qubits))


        for _, edge_list in color_edges_map.items():
            for edge in edge_list:
                qc.append(rzz, edge)


        if barrier:
            qc.barrier()
    return qc
```

In [16]:

```python
def get_circuit_info(qc: QuantumCircuit, reps: int = 0):
    qc0 = qc.decompose(reps=reps)
    return f"{qc0.num_qubits} qubits × {qc0.depth(lambda x: x.operation.num_qubits == 2)} layers ({qc0.c
    ",  " + f"""Gate breakdown: {", ".join([f"{k.upper()} {v}" for k, v in qc0.count_ops().items()])}"""
```

In [17]:

```
backend = fake_provider.FakeTorontoV2()
num_steps = 5
qc = get_utility_circuit(backend, num_steps, True)

display(qc.draw(output="mpl", fold=-1))
print(get_circuit_info(qc, reps=0))
print(get_circuit_info(qc, reps=1))
```


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
q_{14}
q_{15}
q_{16}
q_{17}
q_{18}
q_{19}
q_{20}
q_{21}
q_{22}
q_{23}
q_{24}
q_{25}
q_{26}
RX
RZZ

```
27 qubits × 15 layers (20-depth),  Gate breakdown: CIRCUIT-168 140, RX 135, BARRIER 5
27 qubits × 15 layers (60-depth),  Gate breakdown: SDG 280, UNITARY 280, CX 140, R 135, BARRIER 5
```

In [18]:
```
backend = fake_provider.FakeTorontoV2()    # a 27 qubit fake device.
num_steps = 2
qc = get_utility_circuit(backend, num_steps)
obs = SparsePauliOp.from_sparse_list([("Z", [13], 1)], num_qubits=backend.num_qubits) # Falcon
angles = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, np.pi/2]   # We try 11 angles for theta_h.
```

In [19]:
```
backend_sim = AerSimulator()
transpiled_qc_sim = transpile(qc, backend_sim, optimization_level=1, layout_method="trivial")
transpiled_obs_sim = obs.apply_layout(layout=transpiled_qc_sim.layout)

print(get_circuit_info(qc, reps=1))
print(get_circuit_info(transpiled_qc_sim, reps=1))
```

```
27 qubits × 6 layers (23-depth),  Gate breakdown: SDG 112, UNITARY 112, CX 56, R 54
27 qubits × 6 layers (16-depth),  Gate breakdown: U3 80, CX 56, R 54, U1 32, U 28
```

In [20]:
```
%%time
params = [[p] for p in angles]
estimator = Estimator(mode=backend_sim)
pub = (transpiled_qc_sim, transpiled_obs_sim, params)
result_sim = estimator.run([pub]).result()
```

```
CPU times: user 382 ms, sys: 16.7 ms, total: 399 ms
Wall time: 435 ms
```

In [21]:
```
backend_fake = fake_provider.FakeTorontoV2()
transpiled_qc_fake = transpile(qc, backend_fake, optimization_level=1, layout_method="trivial")
transpiled_obs_fake = obs.apply_layout(layout=transpiled_qc_fake.layout)

print(get_circuit_info(qc, reps=1))
print(get_circuit_info(transpiled_qc_fake, reps=1))
```

```
27 qubits × 6 layers (23-depth),  Gate breakdown: SDG 112, UNITARY 112, CX 56, R 54
27 qubits × 6 layers (49-depth),  Gate breakdown: SDG 324, U1 274, H 162, CX 56, U3 14
```

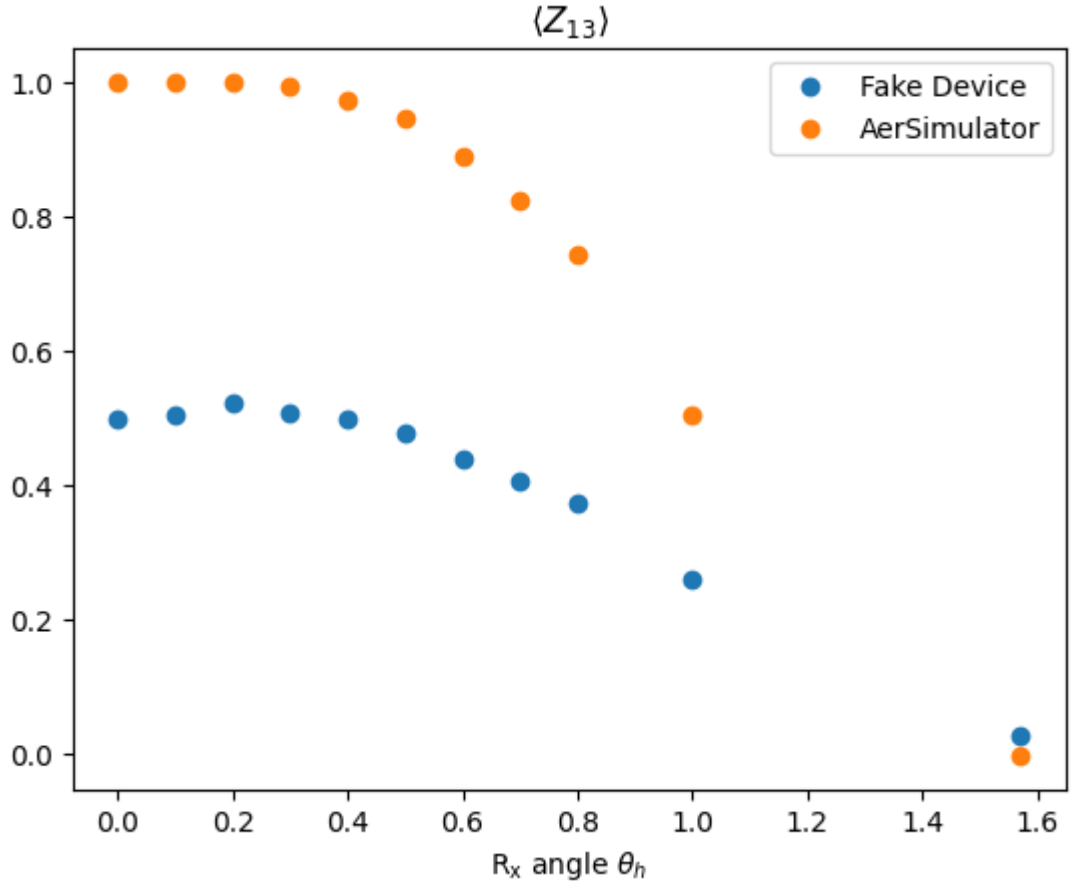
In [22]:

```
%%time
params = [[p] for p in angles]
estimator = Estimator(mode=backend_fake)
pub = (transpiled_qc_fake, transpiled_obs_fake, params)
result_fake = estimator.run([pub]).result()
```

```
CPU times: user 11min 55s, sys: 6.91 s, total: 12min 2s
Wall time: 7min 20s
```

In [23]:

```
1 plt.plot(angles, result_fake[0].data.evs, "o", label="Fake Device")
2 plt.plot(angles, result_sim[0].data.evs, "o", label="AerSimulator")
3 plt.xlabel("$\\mathrm{R_x}$ angle $\\theta_h$")
4 plt.title("$\\langle Z_{13} \\rangle$")
5 plt.legend()
6 plt.show()
```



In [24]:
```
backend_map = service.backend("ibm_torino")

num_steps = 20
qc = get_utility_circuit(backend_map, num_steps)
obs = SparsePauliOp.from_sparse_list([("Z", [62], 1)], num_qubits=backend_map.num_qubits) # Eagle
angles = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 1.0, np.pi/2]
```

In [25]:
```
backend = service.backend("ibm_torino")

transpiled_qc = transpile(qc, backend, optimization_level=1, layout_method="trivial")
transpiled_obs = obs.apply_layout(layout=transpiled_qc.layout)

print(get_circuit_info(qc, reps=1))
print(get_circuit_info(transpiled_qc))
```

```
133 qubits × 60 layers (221-depth),  Gate breakdown: SDG 6000, UNITARY 6000, CX 3000, R 2660
133 qubits × 60 layers (201-depth),  Gate breakdown: RZ 10070, SX 5320, CZ 3000
```

In [26]:
```
params = [[p] for p in angles]
estimator = Estimator(mode=backend)
pub = (transpiled_qc, transpiled_obs, params)
job = estimator.run([pub])

print(f"job id={job.job_id()}")
```

```
job id=d161uhjmk80s73a0oop0
```

In [27]:
```
# REPLACE WITH YOUR OWN JOB IDS
job = service.job("d161uhjmk80s73a0oop0")
```

In [ ]:
```
# get results
result = job.result()
print(result)
```

In [ ]:

```
result_paper = [1.0171, 1.0044, 0.9563, 0.9602, 0.8394, 0.8120, 0.5466, 0.4556, 0.1953, 0.0141, 0.0117]

# REPLACE WITH YOUR OWN JOB ID
job = service.job("d161uhjmk80s73a0oop0")

plt.plot(angles, job.result()[0].data.evs, "o", label=f"{job.backend().name}")
plt.plot(angles, result_paper, "o", label="Utility Paper")
plt.xlabel("$\\mathrm{R_x}$ angle $\\theta_h$")
plt.title("$\\langle Z_{62} \\rangle$")
plt.legend()
plt.show()
```

In [ ]:

```

```