**Why do we care about Quantum?**
Quantum computing is promising in reducing the computation time on certain tasks: e.g., factoring prime numbers (cryptography applications), searching in unstructured data (database), simulating quantum systems (medicine development, new material discovery), or even quantum machine learning (Quantum AI).

**Challenges of Quantum Computing:**
Several applications need a black-box function or a “Quantum Oracle”. The realization of a quantum oracle needs to be carefully handcrafted for different situations. Thus, implementation of a quantum oracle is often difficult and elusive. Further, the oracle needs to be reconfigured for a change in data size, even with the same task. This often hinders the scalability of quantum algorithms.

**Solution we propose:**
*Automatic oracle synthesis* - Given a classical description of the function generate a quantum operation

**Technical background and workflow:**

![Diagram of quantum oracle synthesis into QDK for resource estimation](image)

- **Q#**
  - Classical arithmetic functions/representations (Addition, multiplication, factoring, etc.)
- **C++**
  - Oracle generator
- **QIR file**
  - Function (+, −, ×, ÷ etc.) extraction
- **LLVM**
  - Low-Level Virtual Machine (LLVM) compiler infrastructure is used to build the Oracle generator. Specifically, we read the operators from the classically implemented function and convert it to signals in logic networks.
- **Mockturtle**
  - 1. Logic network synthesis
  - 2. XAG-based optimization
- **QIR file**
  - Quantum logic gates synthesized as a quantum-intermediate representations (QIR) form
- **Verilog**
  - Classical logic network
- **LLVM**
  - Automatically generated Quantum executable
- **Output**
  - Execute with QIR-runner

**Results:**
We successfully implemented functionalities for arithmetic functions: +, −, ×, ÷, etc. Further, we successfully integrate the automatic oracle generation with the Grover search algorithm.

**Realization of classical arithmetic operators**

<table>
<thead>
<tr>
<th>Arithmetic functions</th>
<th>Types</th>
<th>$N_{gates}$</th>
<th>$N_{qubit}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A + B$</td>
<td>64-bit</td>
<td>993 (CX) + 126 (CCX)</td>
<td>319</td>
</tr>
<tr>
<td>$(A + Bx) \mod 11$</td>
<td>64-bit</td>
<td>1360 (CX) + 188 (CCX)</td>
<td>445</td>
</tr>
<tr>
<td>Majority ($A, B, C$)</td>
<td>Bool</td>
<td>10 (CX) + 2 (CCX)</td>
<td>6</td>
</tr>
</tbody>
</table>

**Grover’s Search Algorithm Visualization**
Grover’s algorithm solves a search problem by finding an input $x_0$ that satisfies the condition $f(x_0) = 1$, where $f(x)$ is a classical function mapping $n$-bit search space to $\{0, 1\}$. It’s quantum algorithm provides a quadratic speedup, requiring approximately $\sqrt{N}$ evaluations compared to the classical approach that requires $N$ evaluations, where $N = 2^n$.

**Putting all together:**
Case study: ISBN missing digit search using oracle generator

In the ISBN 10 system, each ISBN is a 10-digit sequence, and the last digit serves as the check. The full sequence $(x_0, x_1, …, x_9)$ should satisfy the following condition: $(\frac{9}{10} - i) x_i$ should be satisfied by the ISBN: $[\text{ISBN}] = [\text{ISBN}] + 9 \cdot 10^8 + 9 \cdot 10^7 + 9 \cdot 10^6 + 9 \cdot 10^5 + 9 \cdot 10^4 + 9 \cdot 10^3 + 9 \cdot 10^2 + 9 \cdot 10^1 + 9 \cdot 10^0$

**Future Work**
After the successful implementation of the Grover’s search algorithm as shown above, we are exploring to implement other algorithms like QFT (Quantum Fourier Transform) and QSVD (Quantum Singular Value Decomposition) using the automatic oracle synthesis code we have developed.