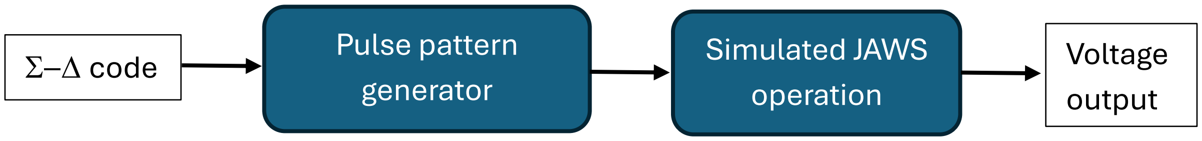
Project title: Simulation of quantum accurate waveform synthesis

Josephson Junctions Arbitrary Waveform Synthesizer (JAWS) is a device constituting superconductors and uses Josephson tunneling, a quantum mechanical phenomenon, to synthesize waveforms with quantum accurate amplitudes.

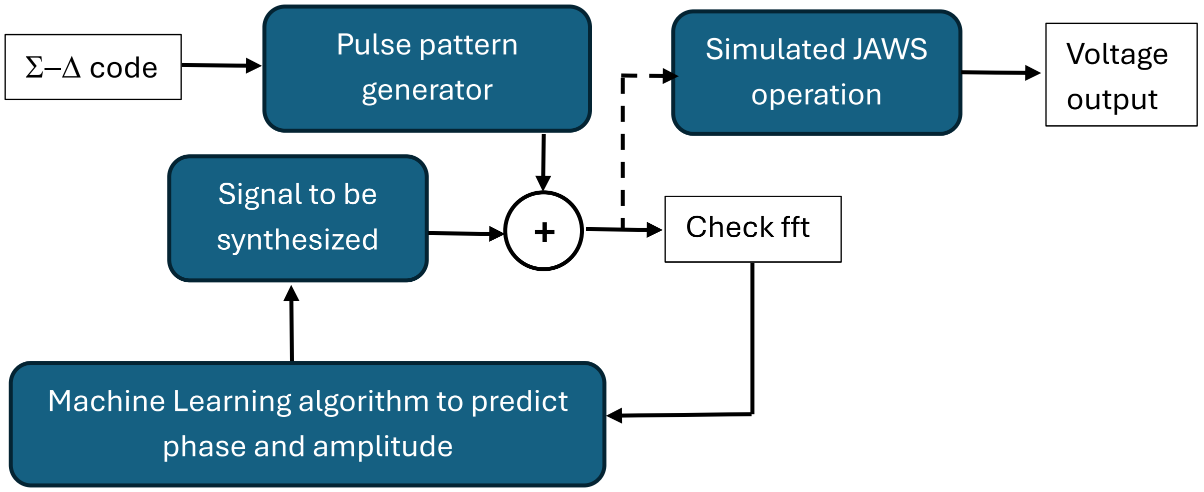
JAWS operation: A sigma-delta code is given to a pulse pattern generator, which generates a very short pulse sequence according to the code. The pulse pattern drives the JAWS device, which demodulates the pulse pattern and generates a voltage output, whose amplitude is proportional to a universal constant.



Problem: Assume that we want to generate a sinusoidal voltage output with a certain frequency *f*. It turns out that the pulse pattern also has a very strong frequency component at *f*. To avoid mixing of the frequency *f* from input to output, we must suppress the frequency *f* in the pulse pattern.

So far, notch band-stop filters have been employed for the task. In this project, we will add a sinusoidal (not quantum accurate) signal with frequency *f* and a certain phase and amplitude to suppress this frequency in the pulse pattern.

Architecture:



A new block for a quantum inaccurate signal generator is added, whose phase and amplitude are decided by a CI/CD machine learning architecture. The generated signal is added to the pulse pattern, and the resulting waveform is Fourier transformed (FFT) to check for the strength of the frequency component at *f*. The strength of *f* is fed back to the machine learning unit for CI, and a new CD is sent to the signal generator unit. The CI/CD process continues until the strength of *f* in the FFT unit, the target variable for the machine learning unit, reduces below a certain threshold. We can turn off the system and restart it with a certain random phase of the pulse pattern and let the machine learning unit continue learning.

Detailed Introduction:

Project Proposal: Smart Signal Synthesis with Machine Learning for JAWS

Overview

The **Josephson Junction Arbitrary Waveform Synthesizer (JAWS)** is a cutting-edge device that generates ultra-precise electrical signals, critical for applications like quantum computing, high-speed communications, and precision medical imaging. JAWS uses a tiny electronic component called a Josephson Junction (JJ) to produce signals with amplitudes tied to universal physical constants, ensuring unmatched accuracy. However, a technical challenge arises: the input pulses that drive JAWS introduce unwanted noise at the same frequency as the desired signal, complicating verification of the output's accuracy.

Traditionally, this noise is filtered out using specialized hardware (notch band-stop filters), which is immutable, complex, and sometime difficult to design. Our project proposes a smarter, software-driven solution: use **machine learning (ML)** to add a carefully tuned counter-signal to the input pulses, canceling the noise without extra hardware. This approach leverages an ML-driven feedback loop, inspired by CI/CD pipelines, to optimize the counter-signal's amplitude and phase, making JAWS more efficient and cost-effective.

Why This Matters

Precise signals are the backbone of transformative technologies:

- **Quantum Computing**: Testing and calibrating quantum processors.
- **Communications**: Enabling faster, more reliable 5G and beyond.
- **Medical Imaging**: Enhancing MRI and other diagnostic tools.

By replacing a static filters with an ML-based solution, we introduce dynamic filtering, simplify deployment, and make JAWS accessible to more industries. For technical teams, this project showcases how ML can solve real-world engineering problems with iterative optimization and scalable software architecture.

The Technical Challenge

JAWS works by feeding a digital sequence, called **sigma-delta code**, into a **pulse pattern generator (PPG)**. This code is like a binary recipe (e.g., 0s, 1s, and -1s) that instructs the PPG to create a series of ultra-fast electrical pulses. These pulses drive the JJ, which converts them into a smooth, precise voltage signal (e.g., a 100 MHz sine wave). The catch? The pulse sequence itself contains a strong frequency component at the same frequency as the desired signal (e.g., 100 MHz in our simulations), which acts like interference in a radio broadcast. This interference makes it hard to confirm the JJ's output is accurate.

Our goal is to suppress this unwanted frequency in the pulse sequence using software, not hardware. We'll achieve this by adding a counter-signal—a sine wave at the same frequency (100 MHz) with optimized amplitude and phase—to cancel the interference, similar to how noise-canceling headphones work.

Our Solution: ML-Driven Signal Optimization

We're building a modular software system that uses ML to automatically find the best counter-signal to suppress the unwanted frequency. The system is designed as a pipeline of interconnected components, with an ML model at its core, optimizing the counter-signal through a feedback loop. Here's how it works at a high level:

1. **Generate the Digital Recipe**: Create the sigma-delta code that defines the desired signal.
2. **Build the Pulse Sequence**: Convert the code into electrical pulses and add a counter-signal.
3. **Analyze the Pulses**: Use a mathematical tool (Fast Fourier Transform, or FFT) to measure the strength of the unwanted frequency.
4. **Optimize with ML**: Adjust the counter-signal's amplitude and phase to minimize the unwanted frequency, using an ML model trained on FFT data.
5. **Test the Output**: Feed the optimized pulses to the JJ and verify the signal's accuracy.

The ML model iterates, testing and refining the counter-signal until the unwanted frequency's strength falls below a threshold (e.g., -80 dB). This process mimics a **CI/CD pipeline**, where the system continuously integrates new pulse data (CI) and deploys updated counter-signal parameters (CD).

System Architecture

The system is built as four modular components, implemented as software containers for scalability and maintainability:

1. **Sigma-Delta Code Generator** (Passive):
    - **Function**: Generates the sigma-delta code, a sequence of bits (e.g., 0, 1, -1) encoding the desired signal (e.g., 1 MHz sine wave).
    - **Input**: Signal parameters (frequency, amplitude) and modulation settings (e.g., 4th-order modulator, 15 GHz sampling).
    - **Output**: A digital sequence, like the 2.4M-bit file used in our simulations.
    - **Tech Notes**: Built in Python using the deltasigma library, streaming output to handle large datasets.
2. **Pulse Pattern Generator (PPG)** (Passive):
    - **Function**: Converts the sigma-delta code into a time-domain pulse sequence and adds a counter-signal (sine wave at 100 MHz).
    - **Input**: Sigma-delta code and counter-signal parameters (amplitude, phase) from the ML model.
    - **Output**: Pulse sequence (e.g., 24,000 cycles of pulses).
    - **Tech Notes**: Implemented in Python with NumPy and Numba for performance, based on our C++ simulation.
3. **Learning Unit** (Active):
    - **Function**: Uses ML to optimize the counter-signal's amplitude and phase, minimizing the 100 MHz component in the pulse sequence's FFT.
    - **Input**: Pulse sequence from PPG and FFT results.
    - **Output**: Updated amplitude and phase parameters.
    - **Tech Notes**:
        - ML Model: To be decided.
        - Loss Function: Power spectral density (PSD) at 100 MHz from the FFT.
        - Feedback Loop: Iterates until the PSD at 100 MHz is below a threshold.
        - Scalability: Processes data in batches (e.g., 100 cycles) with GPU support for FFT and optimization.
4. **JAWS Output Generator** (Passive):

  - **Function**: Simulates the JJ's response to the pulse sequence, producing a voltage signal for verification.
  - **Input**: Pulse sequence from PPG.
  - **Output**: Voltage time series and optional spectrum analysis.
  - **Tech Notes**: Python implementation of the JJ dynamics from our C++ scripts, using Numba for speed.

These components communicate via files or message queues, orchestrated with tools like Docker Compose for container management. The Learning Unit drives the system, using a CI/CD-inspired loop: it analyzes the pulse sequence (CI), updates the counter-signal (CD), and repeats until convergence.

Technical Foundation

Our project builds on existing preliminary attempted simulations developed in C++, which model the JAWS device using a 2.4M-bit sigma-delta code to generate a 1 MHz signal. The C++ code computes the pulse sequence, including a counter-signal (100 MHz sine wave with fixed amplitude and phase), and simulates the JJ's voltage output. We've validated that adding a counter-signal reduces the 100 MHz component in the pulse sequence's spectrum.

The challenge is finding the optimal amplitude and phase for the counter-signal. Our ML approach automates this, replacing manual tuning. The system is designed to handle large datasets (e.g., 24,000 cycles) by streaming data and parallelizing computations, ensuring scalability.

Conclusion

This project transforms JAWS into a smarter, more efficient system by using ML to cancel unwanted noise in the pulse sequence, eliminating the need for costly filters. By combining a modular architecture, a CI/CD-inspired ML pipeline, and scalable software, we're delivering a solution that's both technically robust and practical. For technical teams, this is an exciting opportunity to see ML in action, solving a real engineering problem with precision and innovation.

Authorship and License

Project Proposer: Shekhar Priyadarshi, ORCID iD: 0000-0003-0840-8206

(https://orcid.org/my-orcid?orcid=0000-0003-0840-8206)

Acknowledgments

The author gratefully acknowledges the assistance of Grok 3, created by xAI, in drafting sections of this proposal.