HyPulse: A Pulse Synthesis Framework for Hybrid Qubit-Oscillator Gates on Trapped-Ion Platform

1st Masoud Hakimi Heris
ECE Department
NC State University
Raleigh, NC, USA
mhakimi@ncsu.edu

2nd Yuan Liu
ECE/CS/Physics Departments
NC State University
Raleigh, NC, USA
q_yuanliu@ncsu.edu

3rd Frank Mueller
Computer Science Department
NC State University
Raleigh, NC, USA
fmuelle@ncsu.edu

***Abstract*—As hybrid qubit-oscillator algorithm development and trapped-ion hardware demonstrations advance in parallel, there is a lack of a compilation layer connecting the two at the pulse level in the vertical software stack. While qubit gate control and pulse synthesis are well-established, the translation of hybrid qubit-oscillator primitives to the pulse level has not been systematically addressed. This gap is further compounded by the inherently continuous parametric nature of such gates. Each distinct parameter value defines a physically unique operation requiring independent pulse optimization, making static pre-compilation strategies inapplicable.**

To fill this gap, we present HyPulse, a hardware-aware pulse synthesis and generation framework, which contributes a two-phase architecture decoupling pulse discovery from circuit assembly. An offline optimization engine populates a content-addressed cache of high-fidelity primitives: If a pulse for a given gate, parameter, and device specification already exists in the library, it is retrieved instantly; otherwise the optimizer synthesizes, hashes, and caches it automatically. An online assembler then constructs circuit-specific pulse programs targeting trapped-ion pulse execution backends such as DAX/ARTIQ (Duke) and JaqalPaw/QSCOUT (Sandia). Beyond circuit execution, this architecture enables systematic numerical pre-characterization of parametric gate spaces at software timescales before experimental deployment. We demonstrate the framework through the automated assembly of a squeezed cat state preparation sequence. By reusing cached primitives, HyPulse avoids redundant pulse optimization across circuit instances. The results establish HyPulse as a hardware-portable pulse synthesis and generation infrastructure for hybrid qubit-oscillator trapped-ion processors, with a modular design providing the foundation for further developments including noise-aware gate scheduling, advanced optimal control techniques, and adaptive recalibration. All results reported are closed- and open-system simulations on published hardware parameters; live-hardware validation is future work.

***Index Terms*—Trapped ions, hybrid qubit-oscillator, bosonic qubits, controlled displacement, squeezed cat states, GRAPE, pulse optimization, quantum optimal control, pulse synthesis.**

I. Introduction

Hybrid qubit-oscillator quantum computation encodes information jointly in qubit and oscillator degrees of freedom, leveraging the infinite-dimensional Hilbert space of the oscillator to support richer instruction sets and hardware-efficient error-correcting codes beyond the reach of qubit-only architectures [1]–[3]. Unlike purely discrete-variable processors that operate on a finite register of two-level systems, hybrid processors can exploit the continuous structure of phase space to encode, manipulate, and protect quantum information in ways that are fundamentally inaccessible to qubit-only designs. This distinction has motivated growing interest in hybrid qubit-oscillator systems across many applications. Bosonic encodings such as the Gottesman–Kitaev–Preskill (GKP) code [4] and cat qubit codes [5], [6] encode a logical qubit directly into an oscillator mode, offering hardware-efficient protection against dominant decoherence channels, with recent experimental demonstrations in trapped-ion [7] and superconducting [8] systems confirming their viability as near-term targets. In quantum simulation, the native qubit-oscillator coupling enables direct simulation of vibronic dynamics [9], [10] and spin-boson models [11] that are naturally expressed in a hybrid Hilbert space. In both domains, the required operations are precisely controlled qubit-oscillator interactions, which are the primitives that HyPulse targets.

Recent experimental demonstrations of hybrid oscillator-qubit simulations [10], [11] and a universal gate set for GKP logical qubits [7] on trapped-ion hardware have drawn attention to this platform as a promising substrate for hybrid qubit-oscillator computation. The collective motional modes of a trapped-ion chain are native qumodes that couple directly to the internal qubit states through laser-driven sideband interactions [12]–[14], requiring no additional hardware beyond what is already present in state-of-the-art systems. Crucially, each ion in the chain couples to all motional modes, and each motional mode couples to all ions, providing native all-to-all connectivity between qubit and oscillator degrees of freedom that is intrinsic to the trapping geometry rather than engineered through additional circuitry. This all-to-all structure makes trapped-ion systems a particularly attractive platform for hybrid qubit-oscillator computation, where operations between arbitrary qubit-qumode pairs can be realized without routing overhead.

A full quantum computing stack to support such systems, depicted in Fig. 1, spans four layers: an algorithm layer where circuits are expressed using high-level gate primitives, a compiler and decomposer that transpiles to hardware-native gates, a pulse synthesis layer that converts gate specifications into hardware-ready waveforms, and a hardware execution layer that drives the physical control electronics.

Each layer has seen substantial development. At the algo-

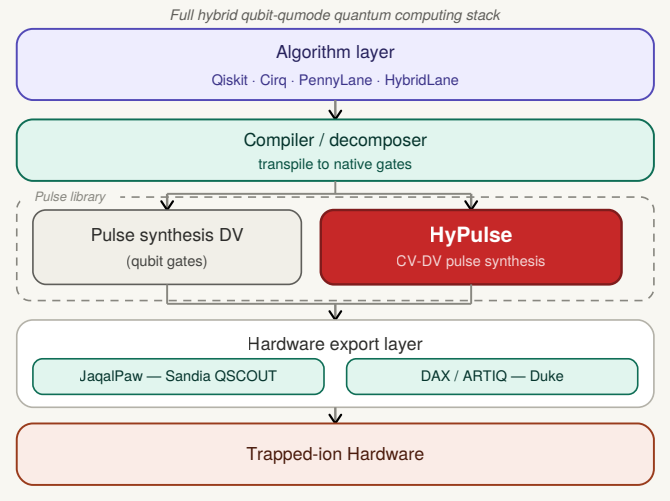

Fig. 1. Full hybrid qubit-qumode quantum computing stack. HyPulse fills the hybrid qubit-qumode pulse synthesis slot between the compiler/decomposer and the hardware export layer, alongside existing DV-only synthesis tools. The content-addressed pulse library is shared between the offline synthesis phase and the online assembly phase.

rithm layer, frameworks such as Qiskit, Cirq, and PennyLane support discrete-variable circuits, while HybridLane [15] extends this to hybrid qubit-oscillator instruction sets [16]. At the hardware execution layer, ARTIQ/DAX [17], [18] and JaqalPaw/QSCOUT [19] serve trapped-ion platforms. QICK [20], designed as a platform-agnostic open-source qubit controller, and QuBIC [21] serve primarily superconducting systems at the same layer. At the pulse synthesis layer, tools such as Qiskit Pulse, PAQOC [22], and EPOC [23] address superconducting DV gates, while Kang et al. [24] and pulselib [25] address trapped-ion DV gates.

The hybrid qubit-oscillator slot of the pulse synthesis layer, however, has no occupant for trapped-ion platforms. No existing tool provides waveform generation for parametric qubit-oscillator gate primitives on trapped-ion hardware with awareness of sideband coupling physics and device calibration. As a result, experimentalists working with hybrid qubit-oscillator circuits on trapped-ion systems must manually run a separate gradient-based optimization for each gate and parameter combination, with no infrastructure for reuse across circuits or sessions, and no systematic way to validate pulse behavior before committing to hardware runs.

Beyond the missing stack layer, hybrid qubit-oscillator gates present a fundamental challenge that has no analogue in the discrete-variable world: They are inherently parametric. A controlled displacement CD(α_1) and CD(α_2) for $\alpha_1 \neq \alpha_2$ are physically distinct operations, each requiring their own pulse optimization run. The same holds for controlled squeezing CS(r) and controlled rotation CR(θ): Every distinct parameter value defines a unique operation demanding a unique optimized pulse.

In the discrete-variable world this problem does not arise because the gate set can be finite. A processor exposes a fixed set of operations that are pre-compiled once during device calibration and reused across all circuits. Even on near-term trapped-ion and superconducting hardware, parametric single-qubit rotations such as Rz(θ) are realized as virtual phase updates in the control software at zero physical cost for any angle [26], requiring no pulse optimization. No analogous trick exists for hybrid qubit-oscillator primitives, which physically transform the oscillator state.

This parametric structure creates a concrete bottleneck in experimental practice. Without dedicated infrastructure, there is no mechanism to reuse optimized pulses across circuits or sessions. Exploring gate behavior over a range of parameter values requires a separate optimization for each point in the sweep, and the cost accumulates directly with the number of parameters explored. As a result, systematic pre-hardware parameter exploration becomes impractical, and experimentalists are forced to commit to hardware runs with limited software-level pre-validation. This is the fundamental bottleneck that motivates the two-phase architecture of HyPulse.

We resolve this gap through HyPulse, a two-phase pulse synthesis and generation framework that decouples the compute-intensive pulse discovery problem from the latency-critical circuit assembly problem.

Phase 1 - Offline Synthesis. A gradient-based optimizer such as GRAPE [27] runs once per gate primitive and parameter combination. Optimized pulses are stored in a content-addressed codebook keyed by a deterministic hash of the gate specification, device calibration, and Hamiltonian model. New parameter combinations trigger synthesis on demand and are cached automatically.

Phase 2 - Online Assembler. Given a hybrid circuit, the assembler resolves each gate to a codebook lookup and stitches the waveforms into a hardware-executable pulse program ready for execution. Phase 1 is always executed first for the target gate and parameter combination: If the pulse already exists in the library it is retrieved instantly; otherwise the optimizer synthesizes, caches, and returns it automatically. Therefore, for circuits composed entirely of previously cached primitives, assembly is dominated by memory lookup and array concatenation, independent of circuit depth. The assembled program is exported directly to DAX/ARTIQ or JaqalPaw for hardware execution.

The remainder of this paper is organized as follows. Section II surveys related pulse-level synthesis frameworks. Section III introduces the three hybrid qubit-oscillator gate primitives targeted by HyPulse. Section IV presents the HyPulse architecture: the four abstraction layers, content-addressed codebook, device calibration layer, compile strategies, and an end-to-end example demonstrating the full pipeline. Section V validates all three gate primitives on published hardware parameters from [7]. Section VI demonstrates multi-gate circuit assembly through a squeezed cat state preparation sequence. Section VII characterizes gate behavior under the hardware-realistic noise model from [7]. Section VIII discusses hardware portability and pre-experimental characterization. Section IX concludes.

II. Related Work

Several prior systems address aspects of pulse-level synthesis for quantum hardware, but none provides a complete path from hybrid qubit-oscillator gate primitives to hardware-ready pulses on trapped-ion systems.

Qiskit Pulse [28] is a pulse-level programming framework implemented within Qiskit-Terra that exposes the OpenPulse interface [29] for superconducting qubit systems. It allows users to define custom pulse schedules and perform Hamiltonian characterization and gate calibration at the pulse level, and has been used to calibrate cross-resonance entangling gates on IBM Quantum hardware. Qiskit Pulse operates on a finite DV gate set for superconducting qubits and has no support for motional modes or hybrid qubit-oscillator primitives.

Boulder Opal [30] is a commercial gradient-based optimal control platform developed by Q-CTRL, providing a flexible optimization engine for arbitrary quantum systems including trapped-ion and superconducting hardware. It supports noise-robust pulse design and has been used in experimental demonstrations of GKP logical qubits [7]. However, Boulder Opal is closed-source and does not provide a built-in content-addressed pulse library tied to device calibration, nor a circuit assembly pipeline targeting trapped-ion control backends such as DAX/ARTIQ and JaqalPaw.

PAQOC [22] proposes a pulse generation framework for superconducting DV circuits that mines frequent subcircuits from quantum programs and converts them into an augmented program-aware basis gate set, using a criticality-based analytical model to prune the optimization search space and reduce compilation overhead. It operates entirely on superconducting DV gates with no representation of motional modes or continuously parametric qubit-oscillator gate specifications.

EPOC [23] proposes an efficient pulse generation framework for superconducting quantum circuits that combines ZX-Calculus, circuit partitioning, and circuit synthesis to accelerate pulse generation and reduce circuit latency by 31.74% compared to prior work [23]. EPOC targets superconducting DV gates and has no support for motional modes or hybrid qubit-oscillator primitives.

Liang et al. [31] propose a hybrid gate-pulse model for variational quantum algorithms on superconducting hardware, separating fixed circuit structures from parameterized layers and applying pulse-level optimization only to the parameterized portions. The two-phase separation of optimization cost from circuit execution is conceptually related to our approach, but their physical model is entirely discrete-variable with no continuous-variable gate support.

pulselib [25] provides a graph-based pulse representation for trapped-ion systems with phase synchronization capabilities, addressing the hardware representation and scheduling layer. It has no optimal control integration and no notion of individual hybrid qudit-qumode gates, operating below the pulse synthesis layer rather than within it.

The Duke group's Mølmer–Sørensen pulse-optimization methods [24] apply pulse optimization to two-qubit gates on trapped-ion hardware. This line of work is the closest to HyPulse in terms of physics-awareness for trapped-ion systems, but operates entirely on the qubit subspace with no support for hybrid qubit–qumode gate primitives.

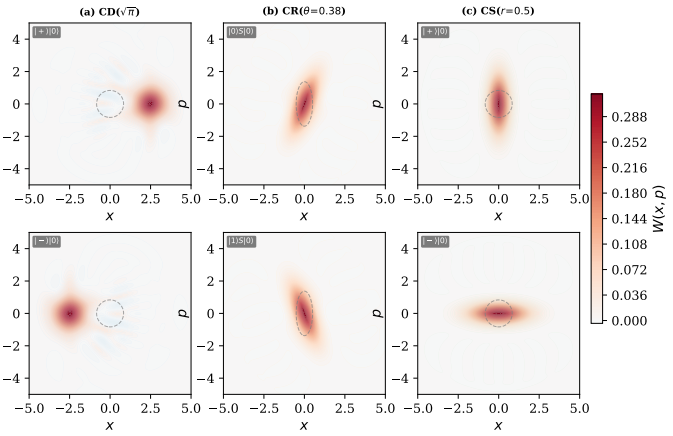

Fig. 2. Phase-space illustration of the three HyPulse gate primitives. Each column shows one gate. Rows show the motional Wigner function conditioned on the two qubit branches, with dashed contours showing the initial state. (a) CD($\sqrt{\pi}$): opposite displacements along x conditioned on $|\pm\rangle$. (b) CR($\theta = 0.38$): opposite phase-space rotations of a squeezed vacuum conditioned on $|0\rangle$, $|1\rangle$. (c) CS($r = 0.5$): opposite squeezings along orthogonal quadratures conditioned on $|\pm\rangle$.

The common gap across all prior work is clear: No existing open-source tool provides systematic pulse synthesis for parametric qubit-oscillator gate primitives on trapped-ion hardware with device-portable calibration and reusable infrastructure. Table I summarizes these characteristics.

TABLE I
COMPARISON OF PULSE-LEVEL SYNTHESIS FRAMEWORKS. "DUKE'S MS" REFERS TO THE MØLMER–SØRENSEN PULSE-OPTIMIZATION METHODS OF KANG ET AL. [24].

Feature	Qiskit Pulse	Boulder Opal	PAQOC	pulselib	Duke's MS	HyPulse
Calibration-aware caching	–	–	–	–	–	✓
Trapped-ion target	–	✓	–	✓	✓	✓
Built-in hybrid gate	–	–	–	–	–	✓
Open-source	✓	–	✓	✓	–	✓
Hardware export	✓	✓	–	✓	–	✓

III. Hybrid Qubit-Qumode Primitive Gates

HyPulse currently targets three fundamental spin–motion gate primitives drawn from the hybrid oscillator-qubit ISA [16], each coupling an ancillary spin qubit to a bosonic motional mode through laser-driven sideband interactions [12], [32]. The framework is designed to accommodate additional primitives as the field develops.

Controlled Displacement (CD). The CD gate is defined as $U_{\mathrm{CD}}(\alpha) = \exp[\sigma_x \otimes (\alpha a^\dagger - \alpha^* a)]$, where $\alpha \in \mathbb{C}$ is the displacement amplitude. In terms of the σ_x eigenstates $|\pm\rangle$:

$$U_{\mathrm{CD}}(\alpha) = |+\rangle\langle+| \otimes D(+\alpha) + |-\rangle\langle-| \otimes D(-\alpha), \quad (1)$$

where $D(\alpha) = e^{\alpha a^\dagger - \alpha^* a}$ is the displacement operator. CD applies opposite phase-space displacements conditioned on the qubit state, and is the elementary operation for cat state generation [8].

Controlled Rotation (CR). The CR gate is defined as $U_{\mathrm{CR}}(\theta) = e^{-i\frac{\theta}{2}\sigma_z \otimes a^\dagger a}$, where θ is the phase-space rotation angle. In terms of the σ_z eigenstates $|0\rangle$, $|1\rangle$:

$$U_{\mathrm{CR}}(\theta) = |0\rangle\langle 0| \otimes e^{-i\theta a^\dagger a} + |1\rangle\langle 1| \otimes e^{+i\theta a^\dagger a}. \quad (2)$$

CR applies opposite rotations in phase space conditioned on the qubit state [16].

Controlled Squeezing (CS). The CS gate is defined as $U_{\mathrm{CS}}(\zeta) = \exp[\frac{1}{2}\sigma_x \otimes (\zeta^* a^2 - \zeta a^{\dagger 2})]$, where $\zeta \in \mathbb{C}$ is the squeezing parameter. In terms of the σ_x eigenstates $|\pm\rangle$:

$$U_{\mathrm{CS}}(\zeta) = |+\rangle\langle+| \otimes S(+\zeta) + |-\rangle\langle-| \otimes S(-\zeta), \quad (3)$$

where $S(\zeta) = \exp[\frac{1}{2}(\zeta^* a^2 - \zeta a^{\dagger 2})]$ is the single-mode squeezing operator. CS drives the second motional sideband and applies opposite squeezings conditioned on the σ_x qubit eigenstates [33].

Fig. 2 illustrates the phase-space effect of each gate, with the two rows showing the motional Wigner function conditioned on the $|+\rangle$ and $|-\rangle$ qubit branches for CD and CS, and the $|0\rangle$ and $|1\rangle$ branches for CR. For CD and CS the initial state is the motional vacuum; for CR a squeezed vacuum is used as the initial state to make the rotation visually apparent. In panel (a), CD displaces the motional vacuum in opposite directions along x depending on the qubit branch, the hallmark of a spin-conditioned force in phase space. In panel (b), CR rotates the squeezed state clockwise or counter-clockwise depending on the qubit branch, turning the elongated Gaussian by $\pm\theta$ around the origin. In panel (c), CS squeezes the motional vacuum along orthogonal quadratures: The $|+\rangle$ branch is squeezed along p while the $|-\rangle$ branch is squeezed along x, producing opposite ellipticities conditioned on the qubit state.

IV. HyPulse Architecture

A. Core Data Structures and Abstraction Layers

HyPulse organizes the pulse synthesis pipeline into four abstraction layers, each hiding complexity from the layers above it, as illustrated in Fig. 3.

Device layer abstracts the physical hardware through `DeviceSpec`, calibration files, and the `CalibrationResolver` protocol. It encodes the Lamb-Dicke parameter η, motional mode frequency ω_m, sideband detuning Δ, maximum drive amplitude $\Omega_{\max}$, participation factors b_{kj}, and coherence properties, shielding all upper layers from hardware-specific physical details.

Gate layer abstracts what to synthesize through `GateSpec`, `ModelSpec`, and the `GatePlugin` protocol.

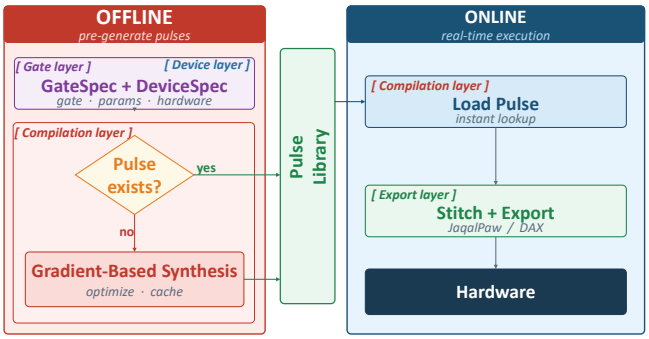

Fig. 3. HyPulse two-phase architecture. The offline phase runs the optimizer once per (gate, device, model) combination and populates a content-addressed library. The online phase assembles circuits from cached pulses, triggering on-demand synthesis only for parameter combinations not yet in the library.

`GateSpec` encodes the gate type (CD, CR, or CS) and its continuous parameter, where α is the displacement amplitude for CD, θ is the phase-space rotation angle for CR, and r is the squeezing parameter for CS. `ModelSpec` encodes the simulation fidelity: Fock space truncation $n_{\max}$, time discretization N_{tslots}, and optimizer convergence tolerances. The `GatePlugin` protocol defines the interface each gate type must implement: parameter validation and canonicalization, codebook key construction, Hamiltonian building, and optimizer invocation. Together these objects specify *what* to build, entirely independent of *how* it is built.

Compilation layer implements the two-phase architecture through `CompiledGate`, `compile_pulse()`, and the SHA-256 content-addressed library. `compile_pulse()` supports three strategies: `lookup` (library only, error on miss), `synthesize` (always re-optimize), and `hybrid` (default: lookup first, synthesize and cache on miss). The `hybrid` strategy ensures the first call for any (gate, device, model) combination pays the synthesis cost once, while all subsequent calls are instant library retrievals. `CompiledGate` is the user-facing lazy handle instantiated via `CompiledGate.cd`, `CompiledGate.cr`, and `CompiledGate.cs`: Calling `cg.pulse()` triggers the full pipeline on demand, and calling `cg.unitary()` additionally propagates the waveform through the Hamiltonian to compute the process unitary.

Export layer translates cached pulses into hardware-ready schedules through `PulseSpec`, `pulse_concat`, `CircuitExporter`, and the DAX/ARTIQ and JaqalPaw backends. It reconstructs the physical laser frequency $\omega_L = \omega_q - \omega_m + \Delta$ from stored device parameters, where ω_q is the qubit transition frequency, ω_m is the motional mode frequency, and Δ is the sideband detuning, and assembles the full pulse sequence with appropriate inter-gate timing buffers. A typical workflow consists of three steps: construct a `CircuitExporter` with the target device and model, add gates via `add_cs()`, `add_cd()`, or `add_cr()`, and call `export()` to produce hardware-ready pulse schedules for both DAX/ARTIQ and JaqalPaw.

The four layers interact strictly through their interfaces: A user touches the gate layer (via `CompiledGate.cd` and similar) and the export layer (via `CircuitExporter`). The device and compilation layers are internal to the framework. Table II summarizes the key components of each layer.

TABLE II
HYPULSE FOUR ABSTRACTION LAYERS AND THEIR KEY COMPONENTS.

Layer	Key Components	Role
Device	`DeviceSpec`, `CalibResolver`	Hardware parameters, mode participation
Gate	`GateSpec`, `ModelSpec`, `GatePlugin`	Synthesis specification
Compilation	`CompiledGate`, `compile_pulse()`, SHA-256	Two-phase caching and strategies
Export	`CircuitExporter`, DAX/JaqalPaw	Hardware-ready pulse schedules

B. Content-Addressed Codebook

The codebook key is computed as `k = SHA256(serialize(G, D, M))`, where `G`, `D`, and `M` are the canonicalized `GateSpec`, `DeviceSpec`, and `ModelSpec`, respectively. Canonicalization ensures deterministic hashing regardless of floating-point representation: All floats are serialized to 17 significant digits to guarantee IEEE 754 double-precision round-trip equality; dictionary keys are sorted lexicographically to produce deterministic JSON regardless of insertion order; complex parameters such as the displacement amplitude $\alpha \in \mathbb{C}$ are decomposed into ordered `(real, imaginary)` tuples to avoid representation ambiguity; and non-finite values (`NaN`, $\pm\infty$) raise an error rather than producing a silently incorrect hash. Together, these rules ensure that two physically identical specifications always produce the same key on any machine, while any physically meaningful change, however small, produces a different key. This design has two important properties beyond performance caching. First, it is collision-resistant under SHA-256: Two physically distinct gate instances map to different keys with cryptographic probability. Second, it is a *physical validity certificate*: If any element of the triple changes, including device calibration drift in η, a shift in ω_m, or a recalibration of $\Omega_{\max}$, the key changes and the stale cached pulse is automatically invalidated without any explicit cache management.

For example, CD($\sqrt{\pi}$) and CD($2\sqrt{\pi}$) on the same device and model produce distinct keys:

```
k1 = _request_hash_for_library(GateSpec("CD", {"alpha_re":
     1.7724, ...}),
                   device, model)
# k1: "3a7f2c..."

k2 = _request_hash_for_library(GateSpec("CD", {"alpha_re":
     3.5449, ...}),
                   device, model)
# k2: "9b1e4d..."  -- distinct, no collision
```

A recalibration that updates η from 0.083 to 0.085 produces a new key for every gate on that device, automatically invalidating all stale cached pulses without any explicit cache management logic.

C. Compile Strategies

The central function of the HyPulse pipeline is `compile_pulse(gate, device, model, strategy)`, which implements the two-phase architecture through three selectable strategies:

lookup: Search the library for an existing pulse matching the (gate, device, model) triple. Return immediately on a hit; raise an error on a miss. Used when synthesis is not permitted at runtime.

synthesize: Always run the optimizer, regardless of whether a cached pulse exists. Used for explicit re-synthesis or benchmarking.

hybrid (default): Attempt a library lookup first. On a cache hit, return the stored `PulseSpec` immediately. On a miss, run the optimizer, hash and save the result to the library under its deterministic path, and return the new pulse. The next call with identical (gate, device, model) finds the stored pulse and skips synthesis entirely.

The `hybrid` strategy is HyPulse's two-phase architecture: The first call for any parameter combination pays the synthesis cost; all subsequent calls are free.

D. User-Facing Interface: CompiledGate

Users interact with HyPulse through the `CompiledGate` object, instantiated via `CompiledGate.cd`, `CompiledGate.cr`, and `CompiledGate.cs`. It stores the gate, device, and model specifications and defers all computation until explicitly requested. On lookup, it first attempts an exact hash match; if not found, it retrieves the highest-fidelity cached pulse for the same gate and device, falling back to synthesis only when no cached pulse exists. This fuzzy fallback enables a common workflow in which Phase 1 synthesizes a pulse with explicit optimizer parameters (`duration_us`, `num_tslots`, `amp_bound`, etc.), while Phase 2 retrieves the cached pulse using only the gate's physics parameters—without needing to repeat the optimizer settings.

Calling `cg.pulse()` triggers the full pipeline, including library lookup and synthesis on a miss, and returns the `PulseSpec`. Calling `cg.unitary()` first calls `cg.pulse()`, then propagates the optimized waveform through the spin-motion Hamiltonian to obtain the process unitary, caching the result separately for subsequent calls.

This lazy evaluation model means that constructing a circuit of N gates incurs no synthesis cost until `pulse()` or `unitary()` is called, and only gates not already in the library trigger new optimization.

E. Device Calibration Layer

HyPulse includes a calibration layer that maps hardware-specific calibration data to the effective Lamb-Dicke parameter

η consumed by the synthesis layer. In a trapped-ion chain, each ion couples to each collective motional mode with a different strength, captured by the mode participation factor $b_{kj} = \eta_{kj}/\eta_k$, where η_k is the single-ion Lamb-Dicke parameter for mode k and b_{kj} is the normalized participation of ion j in that mode [12], [34]. Calibration drift between experimental sessions is handled by the same content-addressed mechanism described in Sec. IV-B. Consider a workflow in which an experimentalist begins a session with a recalibrated device specification `sydney_gkp_2026_04_27` reflecting the latest measured η, ω_m, and $\Omega_{\max}$. Each call to `CompiledGate.cd`, `cr`, or `cs` computes a key over this updated `DeviceSpec`, so any pulse cached under the previous calibration produces a different hash and is automatically bypassed. The optimizer is invoked on demand to populate the new entries, while pulses from prior sessions remain on disk—never served, never deleted—until garbage collection if desired. The user writes no cache-management code; physical validity is enforced by the key construction rather than by runtime checks.

F. Modular Extensibility

HyPulse extends along two independent axes, each requiring minimal new code.

New gate primitives. Adding a gate type requires implementing the `GatePlugin` protocol, which consists of four methods: `validate_and_normalize` (parameter validation and canonicalization), `pulse_lookup_key` (codebook key construction), `synthesize` (optimizer invocation), and `build_hamiltonian` (spin-motion Hamiltonian construction). Once registered via `register(plugin)`, the new primitive is automatically supported by the codebook, online assembler, and hardware export layer without any changes to those components.

```
class CParityPlugin(GatePlugin):
    gate_type = "CParity"

    def validate_and_normalize(self, params):
        # canonicalize phi to [0, 2*pi)
        return {"phi": float(params["phi"]) % (2*
            math.pi)}

    def pulse_lookup_key(self, gate, device, model):
        return canonical_hash(gate, device, model)

    def build_hamiltonian(self, gate, device, model)
        :
        # construct spin-motion H for parity
            interaction
        return H_drift, [H_ctrl_x, H_ctrl_p]

    def synthesize(self, gate, device, model):
        return run_grape(self.build_hamiltonian(...)
            ,
                         target=parity_unitary(gate.
                             phi))

register(CParityPlugin())
```

Listing 1. Minimal `GatePlugin` skeleton for a hypothetical controlled-parity gate.

Listing 1 sketches a minimal plugin definition. The plugin author specifies how parameters are validated, how the codebook key is constructed, and how the spin–motion Hamiltonian is assembled; all caching, scheduling, and hardware export behavior is inherited from the framework.

New hardware targets. Adding a new trapped-ion platform requires only a calibration file specifying η, ω_m, Δ, $\Omega_{\max}$, and coherence properties. If the calibration format matches an existing resolver, no new code is required. If the format is novel, a single resolver class implementing the `CalibrationResolver` protocol is sufficient, and all other framework components remain unchanged. The resolver exposes a uniform interface to the synthesis layer: Given an ion-mode pair, it returns the effective Lamb-Dicke parameter η_{kj} along with any device-specific extras, while the remaining hardware quantities ($\Omega_{\max}$, sideband detuning, and coherence properties) are read directly from the corresponding `DeviceSpec`. This separation enables a single pulse library to span multiple devices—each indexed under its own `DeviceSpec`—without risk of cross-contamination, since the content-addressed key distinguishes pulses by device identity. In practice, retargeting HyPulse from one trapped-ion system to another reduces to writing a small Python calibration module (a few dozen lines defining the `DeviceSpec` and resolver bindings); the gate plugins, optimizer, codebook, and export backends are reused unchanged.

The two axes are orthogonal: A new gate primitive works on all existing hardware targets, and a new hardware target supports all existing gate primitives.

G. Offline Phase: Pulse Synthesis

For each (GateSpec, DeviceSpec, ModelSpec) triple not present in the codebook, HyPulse constructs the appropriate spin-motion Hamiltonian via the registered `GatePlugin` and runs gradient-based optimal control via qutip-qtrl [27], [35]. Amplitude bounds are derived from the device calibration, ensuring the optimized controls remain within hardware limits. The synthesized `PulseSpec` is stored under its deterministic hash path (SHA-256) along with all physical metadata required for hardware export.

H. Online Phase: Circuit Assembly

Given a circuit expressed as a sequence of `CompiledGate` objects, the online assembler:

1) Calls `cg.pulse()` for each gate. Cache hits return immediately; cache misses trigger on-demand synthesis, storage, and retrieval via the `hybrid` strategy.
2) Concatenates piecewise-constant waveforms with appropriate inter-gate timing buffers.
3) Exports the assembled pulse program to DAX/ARTIQ or JaqalPaw, reconstructing the physical laser frequency $\omega_L = \omega_q - \omega_m + \Delta$ from the stored device parameters.

For circuits composed entirely of cached primitives, assembly time is dominated by memory lookup and array concatenation, scaling linearly with gate count, dominated by memory lookup and array concatenation rather than per-gate optimization.

I. End-to-End Example

The following listings illustrate the complete HyPulse pipeline for a CD($0.5+0.5i$) gate on the hardware parameters reported by Matsos et al. [7], demonstrating all four abstraction layers in sequence. Table III summarizes the device parameters used throughout this work. In Phase 1, the gate is synthesized for the first time and cached in the content-addressed library. In Phase 2, the same call returns instantly from cache, the pulse unitary is evaluated, and the circuit is exported to both DAX/ARTIQ and JaqalPaw hardware backends.

Phase 1 — Offline Synthesis.

```
from hypulse.calibrations import get_device_spec
from hypulse.core import CompiledGate
import numpy as np

device = get_device_spec("sydney_gkp_v1")

# Explicit physics parameters for first-time synthesis
cg = CompiledGate.cd(0.5+0.5j,
                     device=device,
                     qubit=0, qumode=0,
                     cutoff=15,
                     duration_us=100.0,
                     num_tslots=200,
                     amp_bound=1.0,
                     Delta_rad_per_s=2*np.pi*10e3,
                     match="exact")
pulse = cg.pulse()  # synthesizes, hashes, saves
```

Phase 2 — Online Assembly.

```
# Cache retrieval
cg = CompiledGate.cd(0.5+0.5j,
                     device=device,
                     qubit=0, qumode=0,
                     cutoff=15)
pulse = cg.pulse()   # cache hit

# Evaluate
U = cg.unitary()
F = process_fidelity(U_target, U)
```

Phase 2 — Hardware Export.

```
# Circuit assembly and hardware export
exporter = CircuitExporter(device, ModelSpec(cutoff=15))
exporter.add_cd(0.5+0.5j)
result = exporter.export("cd_demo",
                         repo_root=Path("."))
# Exports DAX/ARTIQ and JaqalPaw schedules
```

TABLE III
HARDWARE PARAMETERS DERIVED FROM MATSOS ET AL. [7], CORRESPONDING TO THE RADIAL-x MOTIONAL MODE.

Parameter	Value	Description
η	0.083	Lamb-Dicke parameter
ω_m	$2\pi \times 1.33$ MHz	Motional mode frequency
Ω_j	$2\pi \times 2.4$ kHz	Sideband Rabi rate

V. Gate Primitives Validation

We validate all three gate primitives using hardware parameters reported by Matsos et al. [7] for the University of Sydney $^{171}Yb^{+}$ trapped-ion system (referred to as "Sydney" throughout this paper): $\eta = 0.083$, $\omega_m = 2\pi \times 1.33$ MHz, and $\Omega_j = 2\pi \times 2.4$ kHz. Fock space truncation $n_{\max} = 15$ is used throughout. The validation parameters are chosen to exercise each primitive at a physically meaningful operating point: $\alpha = \sqrt{\pi}$ for CD corresponds to the unit cell of the GKP code lattice and the elementary displacement used in cat-state generation; $\theta = 0.382$ for CR is a representative phase-space rotation angle within the linear regime of dispersive coupling; and $r = 0.5$ for CS produces approximately 4.3 dB of squeezing, sufficient to clearly resolve quadrature asymmetry while remaining within the second-sideband coupling regime.

Table IV confirms that all three primitives achieve closed-system fidelity above 0.99 on the mentioned hardware parameters above. CD and CR reach target fidelity within 100–300 µs; CS requires a longer duration of 1300 µs owing to its second-sideband coupling, which accumulates squeezing more slowly than the first-sideband displacement and rotation operations.

Fig. 4 shows the CD gate in detail. Panel (a) presents the optimized pulse waveforms $\Omega_x(t)$ and $\Omega_p(t)$ for $T = 100$ µs, showing the two control channels. Panel (b) shows the Wigner function of the motional mode after applying CD($\sqrt{\pi}$) to the initial state $|+\rangle \otimes |0\rangle$, confirming the expected single coherent lobe at displacement $\alpha = \sqrt{\pi}$. Panel (c) shows the infidelity $1 - \mathcal{F}$ as a function of gate duration T: The operating point $T = 100$ µs lies just below the 10^{-2} infidelity threshold, and longer durations yield further improvement. This duration-fidelity sweep illustrates the pre-hardware characterization capability of HyPulse: Experimentalists can explore the full parameter space in software before committing hardware time.

Optimizer settings: Pulses are synthesized with qutip-qtrl's GRAPE implementation (`optimize_pulse_unitary`), which minimizes the unitary fidelity error via a bounded L-BFGS-B search. Each gate is optimized in a single run from a random initial pulse (`init_pulse_type="RND"`) seeded with a fixed value (`seed` $= 42$), so every synthesis is bit-for-bit reproducible; no multi-start or restart selection is used. The optimizer targets a fidelity-error tolerance of 10^{-5} (10^{-4} for the long controlled-squeezing pulse) with a gradient tolerance of 10^{-20}, a cap of 2×10^4 iterations, and a wall-clock ceiling (1200 s for controlled-displacement and controlled-rotation, shorter for controlled-squeezing); optimization stops at whichever bound is reached first. Control amplitudes are box-constrained to $[-\Omega_{\max}, \Omega_{\max}]$ with $\Omega_{\max}$ fixed per synthesis run.

TABLE IV
GATE PRIMITIVE CLOSED-SYSTEM FIDELITIES ON SYDNEY $^{171}Yb^{+}$ HARDWARE PARAMETERS ($\eta = 0.083$, $n_{\max} = 15$). ALL GATES ACHIEVE $\mathcal{F} \geq 0.99$.

Gate	Parameter	Duration	$\mathcal{F}$
CD	$\alpha = \sqrt{\pi}$	100 µs	0.994
CR	$\theta = 0.382$	300 µs	0.996
CS	$r = 0.5$	1300 µs	0.99999

VI. Squeezed Cat State Demonstration

To demonstrate multi-gate circuit assembly, we prepare a squeezed cat state, a non-classical superposition of two displaced squeezed states whose Wigner function exhibits both spatial separation and quantum interference fringes [36].

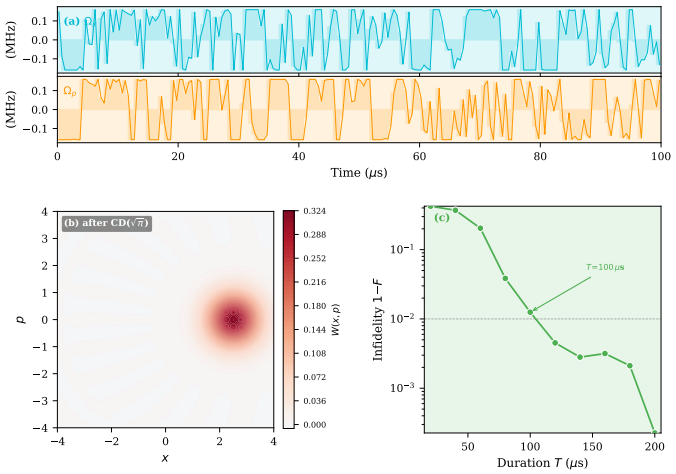

Fig. 4. CD gate characterization on the hardware parameters from Table III. (a) Optimized pulse waveforms $\Omega_x(t)$ (blue) and $\Omega_p(t)$ (orange) for $T = 100$ µs, showing both channels bounded within the hardware amplitude limit. (b) Wigner function of the motional mode after CD($\sqrt{\pi}$), confirming a coherent displacement. (c) Infidelity $1-\mathcal{F}$ vs. gate duration T, demonstrating the pre-characterization sweep capability of the framework.

A. Circuit

The preparation circuit consists of three operations:

1) CS($r = 0.5$) squeezes the motional vacuum along the X quadrature, creating an elongated Gaussian with reduced quantum noise in one direction.
2) This is followed by a single-qubit Hadamard on the spin, rotating from the σ_x to the σ_z eigenbasis.
3) Finally, CD($\alpha = \sqrt{\pi}$) applies a conditional displacement, entangling the spin and the motional mode.

Post-selection on the spin state $|\uparrow\rangle$ then projects the motional mode onto the even squeezed cat state:

$$|\psi_{\text{cat}}\rangle \propto \big(D(\sqrt{\pi}) + D(-\sqrt{\pi})\big) S(0.5)|0\rangle. \tag{4}$$

Both primitives are retrieved from the codebook; no optimization runs at assembly time. This is the first multi-gate circuit assembled by HyPulse and exercises the full pipeline end-to-end: Two distinct gate plugins (CS and CD) contribute pulses synthesized in separate Phase 1 calls, the online assembler concatenates their waveforms with appropriate inter-gate buffers, and the resulting program is exported through the same DAX/ARTIQ and JaqalPaw backends used for single-gate validation. A successful preparation therefore tests not only the individual primitives but also the codebook lookup, waveform stitching, and schedule export logic in a single experiment.

B. Results

Fig. 5 shows the Wigner function at each circuit stage. After CS (panel a), the vacuum Gaussian is elongated along P, confirming squeezing. After CD (panel b), the state is entangled: Two separated squeezed lobes are visible, corresponding to the two spin-conditioned displacement branches. After post-selection on $|\uparrow\rangle$ (panel c), the motional mode is projected

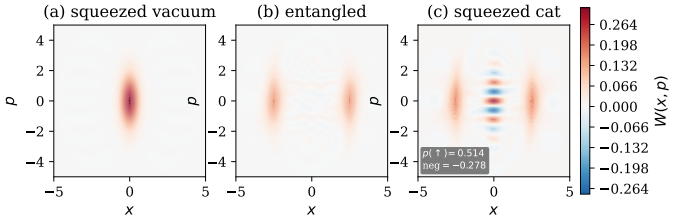

Fig. 5. Wigner function evolution through the squeezed cat preparation circuit on Sydney hardware parameters. (a) After CS($r = 0.5$): squeezed vacuum. (b) After CS + CD($\sqrt{\pi}$): spin-motion entangled state with two separated lobes. (c) After post-selection on $|\uparrow\rangle$: squeezed cat state with Wigner negativity neg $= -0.278$, confirming non-classical coherence consistent with Kienzler et al. [36].

onto the squeezed cat state with peak Wigner negativity neg $= -0.278$ and post-selection probability $p(\uparrow) = 0.514 \approx 0.5$, consistent with the expected even-parity cat state.

The clear Wigner negativity in panel (c) is the definitive signature of non-classical coherence, confirming that the two HyPulse-compiled primitives compose correctly without introducing phase errors or timing artifacts at the stitch boundary.

VII. Noise Characterization

A primary use case of HyPulse is systematic numerical pre-characterization of gate behavior under hardware-realistic noise before experimental deployment. We simulate each primitive under the motional dephasing model reported by Matsos et al. [7]: The noisy Hamiltonian includes a dephasing term $H_{\text{deph}}(t) = \epsilon(t)\, a^\dagger a$, where $\epsilon(t)$ is a piecewise-constant Gaussian process with variance $\sigma^2 = 2\gamma/\tau_{\text{seg}}$, decay rate $\gamma = 18$ Hz, and segment duration τ_{seg} matching the pulse time discretization. Results are averaged over 500 independent noise trajectories.

TABLE V
Gate fidelities under Sydney's motional dephasing model ($\gamma = 18$ Hz, 500 trajectories).

Gate	Duration	$\mathcal{F}_{\text{closed}}$	$\mathcal{F}_{\text{noisy}}$	$\Delta\mathcal{F}$
CD	100 µs	0.994	0.922 ± 0.04	0.07
CR	300 µs	0.996	0.850 ± 0.18	0.15
CS	1300 µs	0.99999	0.554 ± 0.33	0.45

Table V and Fig. 6 summarize the results. CD and CR are less affected by noise at current parameters, with fidelity degradation of 7% and 15%, respectively, consistent with their gate durations of 100 µs and 300 µs under the $\gamma = 18$ Hz dephasing rate. CS is noise-limited: Its second-sideband coupling requires 1300 µs to accumulate $r = 0.5$ squeezing, and dephasing over this duration introduces $\Delta\mathcal{F} \approx 0.45$ with high variance, indicating sensitivity to the noise realization. This reflects current hardware constraints, not an architectural limitation of HyPulse. Improved sideband Rabi rates on future hardware would directly reduce the CS gate time and suppress the noise floor. Identifying this constraint in software, before any trap time is committed, illustrates HyPulse's role as a pre-experimental screen: The framework makes it possible

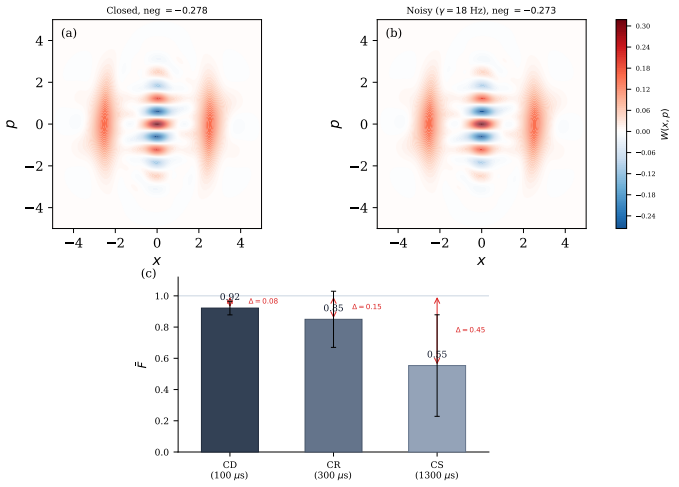

Fig. 6. Noise characterization under the motional dephasing model reported in Matsos et al. [7] ($\gamma = 18$ Hz, 500 trajectories). (a) Closed-system squeezed cat Wigner function, $\mathrm{neg} = -0.278$. (b) Noisy Wigner function averaged over 500 trajectories, $\mathrm{neg} = -0.273$: negativity preserved. (c) Per-gate noisy fidelity $\mathcal{F}_{\mathrm{noisy}}$ for CD, CR, and CS. CD and CR are less affected by noise; CS is limited by its 1300 µs gate duration, a hardware constraint addressable by improved sideband Rabi rates.

to quantify the gap between current and required hardware performance ahead of deployment, and to translate noise budgets directly into hardware development targets such as required sideband Rabi rates and motional coherence times.

Crucially, the squeezed cat circuit preserves Wigner negativity under noise. The peak negativity degrades only from -0.278 to -0.273, a 1.8% reduction that leaves the interference fringes clearly visible. The post-selection on $|\uparrow\rangle$ projects the motional mode onto the cat superposition, and the modest per-lobe squeezing reduction under dephasing does not destroy the interference structure responsible for Wigner negativity.

VIII. Discussion

Physical validity as a structural property. The content-addressed key makes physical validity a structural property of the cache rather than a runtime check. A pulse synthesized for $\eta = 0.083$ is never served to a circuit targeting $\eta = 0.10$; and a pulse cached before a trap recalibration is never reused after. This property is especially important for long-running experimental campaigns where device parameters drift between sessions, and removes an entire class of subtle correctness bugs from the user's responsibility.

Hardware portability. Retargeting HyPulse to a new trapped-ion platform requires only a new calibration file. The framework supports the hardware reported in [7], producing distinct pulse libraries that correctly reflect the different physical coupling strengths and mode structures.

Pre-experimental characterization as a first-class use case. The duration-fidelity sweep of Fig. 4(c) and the noise characterization of Sec. VII illustrate HyPulse's role as a pre-experimental exploration environment. Experimentalists can identify viable operating points, assess noise sensitivity, and project the impact of hardware improvements entirely in software before committing trap time. The CS noise results in particular quantify how improved sideband Rabi rates would reduce gate duration and suppress the noise floor, providing a concrete software-derived target for hardware development.

Limitations and threats to validity. The validation in this paper is entirely numerical: Pulses are synthesized and characterized in simulation against the device parameters reported in [7], and the noise model is restricted to the dominant motional dephasing channel. Hardware-specific effects such as cross-mode coupling, AC Stark shifts, laser intensity noise, and finite spin coherence are not included in the present analysis and would shift the achievable fidelities in practice. Validation against live hardware data, integration with additional trapped-ion platforms beyond the reference parameters, and extension of the noise model to incorporate vendor-supplied noise spectra are the natural next steps and are scoped as future work in Sec. X. The noise study is restricted to motional dephasing. Other channels such as motional heating, mode crosstalk, laser intensity noise, and laser phase noise are not modeled. Similarly, portability is demonstrated on a single set of published device parameters. Performance across substantially different trap geometries or calibration regimes has not been evaluated. Both extensions are natural next steps and are compatible with the plugin and calibration interfaces described in Sec. IV.

IX. Conclusion

We presented **HyPulse**, an open-source two-phase pulse synthesis framework that fills the gap in the vertical quantum stack of hybrid qubit-oscillator trapped-ion architectures. The framework addresses the fundamental challenge of continuously parametric gates: By separating the one-time offline synthesis cost from repeated online circuit assembly, HyPulse provides the infrastructure for systematic pulse reuse, hardware-portable calibration, and pre-experimental parameter exploration that is currently absent for hybrid qubit-oscillator systems.

Validated on the hardware parameters from [7], HyPulse achieves closed-system fidelity above 0.99 for all three primitives and assembles a squeezed cat state preparation circuit with preserved Wigner negativity ($\mathrm{neg} = -0.273$) under the reported dephasing model in [7]. CD and CR are less affected by noise at current hardware parameters. CS noise sensitivity is tied to its longer gate duration, a hardware constraint that improved sideband Rabi rates would directly address.

The modular architecture of HyPulse is designed to grow with the field. As new hybrid gate primitives are proposed and new trapped-ion platforms come online, the framework absorbs them through its plugin and calibration interfaces. As optimization techniques mature, they plug directly into the offline synthesis phase without affecting the rest of the stack.

Together, these properties position HyPulse as a foundation for systematic, reusable, and hardware-aware pulse synthesis in the hybrid qubit-oscillator stack, complementing existing discrete-variable tooling and lowering the barrier to experi-

mental exploration of bosonic quantum information processing on trapped-ion devices.

X. Future works

Future directions include noise-aware pulse synthesis that incorporates hardware noise spectra directly into the optimization objective, multi-gate joint optimization that exploits pulse overlap between adjacent gates, and extension of the gate primitive set to include two-qubit hybrid entangling operations and controlled parity gates. On the hardware side, integration with additional trapped-ion platforms and validation against experimental data from live hardware runs remain important next steps.

Beyond these, several directions follow naturally from the two-phase architecture. Adaptive recalibration could close the loop between cached pulses and live device feedback, automatically re-synthesizing primitives when measured fidelity drops below a threshold. The content-addressed library is also a natural target for transfer-learning approaches, in which previously optimized pulses on one device serve as warm starts for synthesis on a related device. Finally, integrating HyPulse with higher-level hybrid compilers such as HybridLane [15] would enable end-to-end compilation from algorithm-level circuits to hardware-ready pulse schedules, completing the vertical stack sketched in Fig. 1.

XI. Code and Data Availability

HyPulse is open-source under the MIT License at https://github.com/fmuelle4711/hypulse. The repository includes the implementation, calibration files, and scripts to reproduce all figures and tables in this paper.

Acknowledgments

This work was supported in part by National Science Foundation grants OSI-2531350, PHY-2325080a and OMA-2120757. Yuan Liu and Frank Mueller were also supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under contract number DE-SC0025384.

References

[1] U. L. Andersen, J. S. Neergaard-Nielsen, P. van Loock, and A. Furusawa, "Hybrid discrete- and continuous-variable quantum information," *Nature Physics*, vol. 11, pp. 713–719, 2015.

[2] S. Lloyd and S. L. Braunstein, "Quantum computation over continuous variables," *Phys. Rev. Lett.*, vol. 82, pp. 1784–1787, 1999.

[3] C. Weedbrook, S. Pirandola, R. García-Patrón, N. J. Cerf, T. C. Ralph, J. H. Shapiro, and S. Lloyd, "Gaussian quantum information," *Rev. Mod. Phys.*, vol. 84, pp. 621–669, 2012.

[4] D. Gottesman, A. Kitaev, and J. Preskill, "Encoding a qubit in an oscillator," *Phys. Rev. A*, vol. 64, p. 012310, 2001.

[5] M. Mirrahimi, Z. Leghtas, V. V. Albert *et al.*, "Dynamically protected cat-qubits: a new paradigm for universal quantum computation," *New J. Phys.*, vol. 16, p. 045014, 2014.

[6] A. L. Grimsmo, J. Combes, and B. Q. Baragiola, "Quantum computing with rotation-symmetric bosonic codes," *PRX Quantum*, vol. 1, p. 010321, 2020.

[7] V. G. Matsos, C. H. Valahu, M. J. Millican, T. Navickas, X. C. Kolesnikow, M. J. Biercuk, and T. R. Tan, "Universal quantum gate set for Gottesman–Kitaev–Preskill logical qubits," *Nature Physics*, 2025.

[8] A. Eickbusch, V. Sivak, A. Z. Ding *et al.*, "Fast universal control of an oscillator with weak dispersive coupling to a qubit," *Nature Physics*, vol. 18, pp. 1464–1469, 2022.

[9] R. J. MacDonell, T. Navickas, D. Wecker *et al.*, "Analog quantum simulation of chemical dynamics," *Chem. Sci.*, vol. 12, pp. 9794–9805, 2021.

[10] E. Crane, K. C. Smith, T. Tomesh, A. Eickbusch, J. M. Martyn, S. Kühn, L. Funcke, M. A. DeMarco, I. L. Chuang, N. Wiebe, A. Schuckert, and S. M. Girvin, "Hybrid oscillator-qubit quantum processors: Simulating fermions, bosons, and gauge fields," *arXiv preprint arXiv:2409.03747*, 2024.

[11] J. Y. Araz, M. Grau, J. Montgomery, and F. Ringer, "Hybrid quantum simulations with qubits and qumodes on trapped-ion platforms," *Phys. Rev. A*, vol. 112, p. 012620, 2025.

[12] D. Leibfried, R. Blatt, C. Monroe, and D. Wineland, "Quantum dynamics of single trapped ions," *Rev. Mod. Phys.*, vol. 75, p. 281, 2003.

[13] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 1995.

[14] H. Häffner, C. F. Roos, and R. Blatt, "Quantum computing with trapped ions," *Phys. Rep.*, vol. 469, pp. 155–203, 2008.

[15] J. Furches *et al.*, "Hybridlane: A software development kit for hybrid continuous-discrete variable quantum computing," *arXiv preprint arXiv:2603.10919*, 2026.

[16] Y. Liu, S. Singh, K. C. Smith *et al.*, "Hybrid oscillator–qubit quantum processors: Instruction set architectures, abstract machine models, and applications," *PRX Quantum*, vol. 7, p. 010201, 2026.

[17] M-Labs, "ARTIQ: Advanced real-time infrastructure for quantum physics," https://m-labs.hk/artiq, 2014.

[18] L. Riesebos, B. Bondurant, J. Whitlow, J. Kim, M. Kuzyk, T. Chen, S. Phiri, Y. Wang, C. Fang, A. Van Horn, J. Kim, and K. R. Brown, "Modular software for real-time quantum control systems," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 545–555.

[19] S. M. Clark, D. Lobser, M. C. Revelle *et al.*, "Engineering the Quantum Scientific Computing Open User Testbed (QSCOUT): Design details and user guide," *arXiv preprint arXiv:2104.00759*, 2021.

[20] L. Stefanazzi, K. Treptow, N. Wilcer *et al.*, "The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors," *Rev. Sci. Instrum.*, vol. 93, p. 044709, 2022.

[21] Y. Xu, G. Huang, J. Balewski *et al.*, "QubiC: An open-source FPGA-based control and measurement system for superconducting quantum information processors," *IEEE Trans. Quantum Eng.*, vol. 2, pp. 1–11, 2021.

[22] Y. Chen, Y. Jin, F. Hua, A. Hayes, A. Li, Y. Shi, and E. Z. Zhang, "A pulse generation framework with augmented program-aware basis gates and criticality analysis," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 773–786.

[23] J. Cheng, Y. Zhu, Y. Zhou, H. Ren, Z. Song, and Z. Liang, "EPOC: An efficient pulse generation framework with advanced synthesis for quantum circuits," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.

[24] M. Kang, Q. Liang, B. Zhang, S. Huang, Y. Wang, C. Fang, J. Kim, and K. R. Brown, "Batch optimization of frequency-modulated pulses for robust two-qubit gates in ion chains," *Phys. Rev. Applied*, vol. 16, no. 2, p. 024039, 2021.

[25] A. S. Dalvi, L. Riesebos, J. Whitlow, and K. R. Brown, "Graph-based pulse representation for diverse quantum control hardware," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2024, pp. 897–908.

[26] D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, and J. M. Gambetta, "Efficient Z gates for quantum computing," *Phys. Rev. A*, vol. 96, p. 022330, 2017.

[27] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, "Optimal control of coupled spin dynamics: Design of NMR pulse sequences by gradient ascent algorithms," *J. Magn. Reson.*, vol. 172, pp. 296–305, 2005.

[28] T. Alexander, N. Kanazawa, D. J. Egger *et al.*, "Qiskit Pulse: Programming quantum computers through the cloud with pulses," *Quantum Sci. Technol.*, vol. 5, p. 044006, 2020.

[29] D. C. McKay, T. Alexander, L. Bello *et al.*, "Qiskit backend specifications for OpenQASM and OpenPulse experiments," *arXiv preprint arXiv:1809.03452*, 2018.

[30] Q-CTRL, "Boulder Opal: Quantum control infrastructure software," https://q-ctrl.com/boulder-opal, 2021.
[31] Z. Liang, Z. Song, J. Cheng, Z. He, J. Liu, H. Wang, R. Qin, Y. Wang, S. Han, X. Qian, and Y. Shi, "Hybrid gate-pulse model for variational quantum algorithms," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.
[32] D. J. Wineland, C. Monroe, W. M. Itano, D. Leibfried, B. E. King, and D. M. Meekhof, "Experimental issues in coherent quantum-state manipulation of trapped atomic ions," *J. Res. Natl. Inst. Stand. Technol.*, vol. 103, pp. 259–328, 1998.
[33] O. Katz, M. Cetina, and C. Monroe, "Programmable N-body interactions with trapped ions," *PRX Quantum*, vol. 4, p. 030311, 2023.
[34] K. Sun, M. Kang, H. Nuomin, G. Schwartz, D. N. Beratan, K. R. Brown, and J. Kim, "Quantum simulation of spin-boson models with structured bath," *Nat. Commun.*, vol. 16, p. 4042, 2025.
[35] J. R. Johansson, P. D. Nation, and F. Nori, "QuTiP: An open-source Python framework for the dynamics of open quantum systems," pp. 1760–1772, 2012.
[36] D. Kienzler, H.-Y. Lo, B. Keitch, L. de Clercq, F. Leupold, F. Lindenfelser, M. Marinelli, V. Negnevitsky, and J. P. Home, "Quantum harmonic oscillator state synthesis by reservoir engineering," *Science*, vol. 347, pp. 53–56, 2015.