UCLouvain

École polytechnique de Louvain

Comparative Analysis of Direct and Indirect Coupling in Superconducting Two-Qubit Circuits

Author: **Aurélien Loog**
Supervisor: **Sorin Melinte**
Readers: **Renan Villareal de la Fuente, Matthieu Génévriez**
Academic year 2025–2026
Master [120] in Electrical Engineering

Abstract

In a superconducting processor, a coupler is the circuit element through which two qubits exchange excitations. It sets the interaction strength g, hence the rate at which the two qubits exchange excitations and the timescale on which a two-qubit gate can be performed: a gate must be fast compared to the coherence time for its error to stay low, so a strong coupling is wanted, up to the point where the residual interaction it leaves on between gates becomes the limiting factor. A coupler may be a capacitance drawn directly between the two qubit islands, a resonator shared by both and detuned from them, or a tunable element switching the interaction on and off. The choice is a determining factor for processor performance, because the same metal that produces the coupling also loads the qubits it connects, shifting the frequency and the anharmonicity they were designed for.

Starting from Yohan Burignat's Master thesis, which designed and validated a single-transmon chip, this work extends that layout to two qubits, a transmon pocket and an Xmon, each carrying its own quarter-wave readout resonator and charge line. Two configurations are compared: a reference layout with no coupler, in which the qubits interact only through the shared readout network, and a direct capacitive coupler formed by extending one Xmon arm until it faces a capacitor pad of the pocket. Both are built in *Qiskit Metal* and simulated by finite elements in the Ansys Quasi-static 3D Extractor, whose capacitance matrix feeds a two-qubit lumped oscillator quantisation, and in the Ansys High-Frequency Structure Simulator, whose eigenmodes are post-processed through energy participation ratios. No parameter is fitted, and the two branches agree to better than 7 % on the frequencies and anharmonicities of the converged modes.

The direct coupler yields $g_{12}/2\pi = 25.9$ MHz, a coupling rate corresponding to about 10 ns, against 0.034 MHz for the reference layout, which corresponds to some 7 µs and leaves no usable interaction. The coupler therefore does its job, but not for free. It leaves the transmon pocket untouched, its island capacitance moving by 0.5 %, while tripling that of the Xmon, from 111.7 to 374.2 fF, driving the ratio of Josephson to charging energy E_J/E_C from 94 to 316 and the qubit frequency from 4.6 down to 2.6 GHz, 3.3 GHz below its neighbour, a detuning large enough for the interaction to be suppressed rather than used. Only 5.63 fF of the 262 fF added actually face the neighbouring qubit, so this penalty follows from the geometry of the coupler rather than from the coupling itself. A direct coupler cannot be inserted into an already validated single-qubit layout without re-optimising the qubit that carries it.

AI Tool Usage Disclaimer

In preparing this Master's thesis, the AI-based language model (*Claude*, Anthropic) was employed to enhance clarity, style, and grammatical consistency, and to assist with parts of the code implementation: specifically the writing of methods extending beyond *qiskit-metal*'s built-in functions, and the LOM implementation. All AI-assisted code was reviewed, tested, and validated by the author. All substantive research activities, including data collection and analysis, and interpretation of results, were conducted and validated by the author. The author assumes full responsibility for the accuracy and integrity of this document.

Remerciements

Ce travail de fin d'études marque le terme de ma formation universitaire. Je voudrais remercier Sorin et Renan pour leur encadrement et leur investissement : les réunions, les appels et tout le temps consacré à relire mon manuscrit. Merci également à mes parents et à Flore, pour leur soutien et pour avoir toujours cru en moi, ainsi qu'à toutes les personnes rencontrées au cours de ces années. Je m'y suis fait d'excellents amis, grâce à qui l'université aura tenu la promesse de son nom, au lieu de se réduire à l'assimilation d'une série de cours techniques.

À Ira, qui aurait aimé lire ces pages.

Contents

Glossary

Term	Definition
	Acronyms
CNOT	Controlled-NOT
CPW	Coplanar waveguide
cQED	Circuit quantum electrodynamics
CZ	Controlled-Z
DFS	Decoherence-free subspace
DM	Dzyaloshinskii–Moriya (interaction)
EPR	Energy participation ratio
FPGA	Field programmable gate array
GDSII	Graphic Design System II (GDS) layout file format
HFSS	High-Frequency Structure Simulator
IBM	International Business Machines Corporation
iSWAP	Imaginary-SWAP two-qubit gate
JJ	Josephson junction
LC	Inductor-capacitor
LOM	Lumped oscillator model
NN	Nearest-neighbor
NNN	Next-nearest-neighbor
Q3D	Quasi-static 3D extractor
QEC	Quantum error correction
QPU	Quantum processing unit
RSA	Rivest–Shamir–Adleman
RWA	Rotating-wave approximation
SQUID	Superconducting Quantum Interference Device
SW	Schrieffer-Wolff
SWT	Schrieffer-Wolff transformation
SWAP	SWAP two-qubit gate
Syndrome	Bit string formed by stabilizer measurement outcomes
TL	Transmission line
Transmon	Transmission-line shunted plasma-oscillation qubit
Xmon (XMON)	Cross-shaped transmon
ZPF	Zero-point fluctuation

Term	Definition
Greek Symbols	
α	Anharmonicity
Γ_1	Energy relaxation rate
Δ_j	Qubit-coupler detuning
Δ_{12}	Qubit-qubit detuning
λ	Wavelength
$\hat{\sigma}^{\pm}$	Pauli raising / lowering operators
$\hat{\sigma}_j^z$	Pauli-Z operator acting on qubit j
Φ_0	Superconducting flux quantum
Φ_{ext}	External magnetic flux through the SQUID loop
Φ_i	Node flux at circuit node i
ω_c	Coupler frequency (tunable)
ω_c^{off}	Coupler frequency at which the effective qubit-qubit coupling vanishes
ω_i	Angular frequency of qubit i
ω_q	Qubit transition (angular) frequency
ω_r	Resonator resonance (angular) frequency
ω_{01}	Qubit ground-to-first-excited transition frequency
Other Symbols	
$\hat{a}_i, \hat{a}_i^\dagger$	Annihilation / creation operator of qubit i
C_{12}	Direct coupling capacitance between the two qubits
$C_{d,i}$	Drive (charge-line) coupling capacitance of qubit i
C_{env}	Stray capacitance to the surrounding environment
C_g	Coupling capacitance
$C_{g,j}$	Qubit-coupler coupling capacitance
C_J	Josephson junction (intrinsic) capacitance
$C_{k,i}$	Feedline coupling capacitance of resonator i
C_r	Resonator capacitance
C_S	Shunt capacitance of the transmon
C_Σ	Total island capacitance
d	SQUID junction asymmetry
E_C	Charging energy of the transmon island
E_J	Josephson energy of a single junction

Term	Definition
E_J^{eff}	Effective Josephson energy of the SQUID
$E_{J\Sigma}$	Sum of the two SQUID junction energies
e	Elementary charge
g	Qubit-qubit coupling constant
g_{12}	Direct (NNN) qubit-qubit coupling strength
g_j	Qubit-coupler coupling strength
$\tilde{g}$	Effective qubit-qubit coupling after Schrieffer-Wolff transformation
$\hat{H}$	Hamiltonian
$\hbar$	Reduced Planck constant
I_c	Critical current of a Josephson junction
$\|0\rangle, \|1\rangle$	Qubit computational (ground / excited) states
$\|0\rangle_L, \|1\rangle_L$	Logical qubit computational states
$\|\psi\rangle$	Qubit pure state
L_J	Josephson inductance
L_r	Resonator inductance
p_{diel}	Substrate dielectric participation ratio
$p_{jj,i}$	Junction participation ratio of junction i
Q_i	Node charge at circuit node i
$Q_{\text{ZPF},i}$	Zero-point charge fluctuation of qubit i
R_i	Register (ancilla measurement block)
T_1	Energy relaxation time
T_{gate}	Characteristic single- or two-qubit gate time
$V_{d,i}(t)$	Microwave drive voltage source on qubit i
Z_0	Characteristic impedance of a transmission line

1 Introduction

1.1 Typical quantum computing system

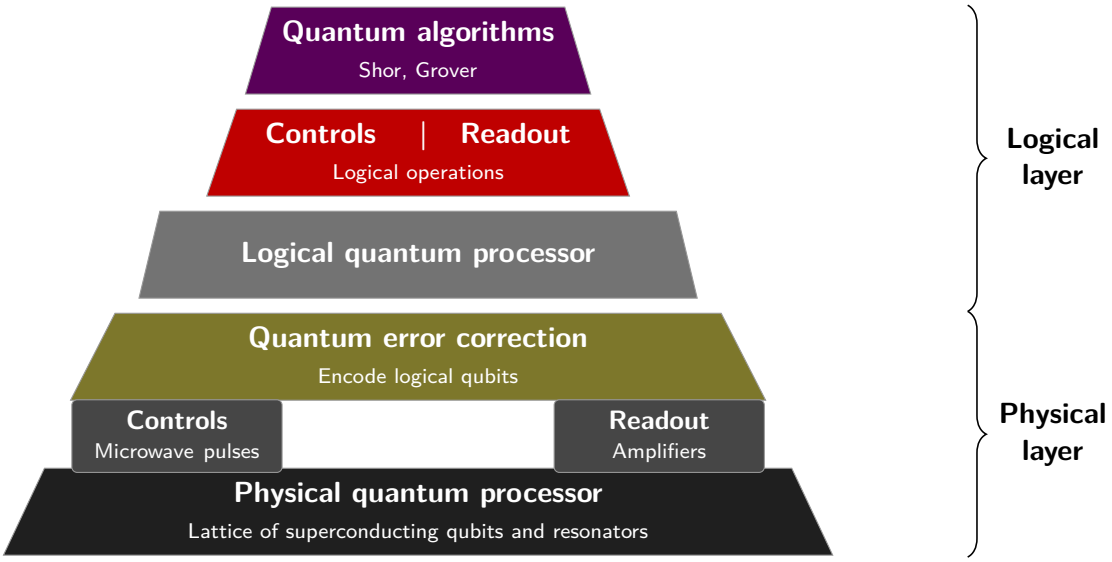


Figure 1: A systems view of a quantum information processor. It consists of a physical and logical layer. The physical layer provides the error correction and consists of a physical quantum processor that has both input and output lines that are controlled by the QEC processor. This processor is in turn controlled by the logical layer, where the encoded qubits are defined and the logical operations are performed for the desired quantum algorithm. Inspired from Ref. [1].

The idea of quantum computing is to use quantum mechanics to store information in the values of 2^N complex amplitudes describing the wavefunction of N two-level systems (qubits), and to process this information by applying unitary transformations (quantum gates) that change these amplitudes in a precise and controlled manner. The number of logical qubits (Sec. 1.1.1) needed to run a practically useful algorithm is estimated to be on the order of 10^3 or more.

A full fault-tolerant quantum computing system can be envisioned within a layered structure as shown in Fig. 1. The system is comprised of two primary layers, a physical qubit layer and a logical qubit layer. The lower physical layer contains physical qubits controlled via a quantum error correction (QEC)[1] processor (Sec. 1.1.3), which is in essence a classical processor that uses measurement outcomes of the physical qubits to realize a QEC code. This classical processor keeps track of the physical errors that arise, and implements the appropriate feedback on the controls of the physical qubits. The upper layer of Fig. 1 is called a logical layer and functions through control of the physical layer. Here, logical qubits are encoded within the fully error-corrected system of physical qubits, and logical controls and readouts are governed through a processor that determines

[1] Physical qubits suffer from decoherence (loss of quantum information to the environment) and imperfect gates, producing a range of errors that have no classical analog: a classical bit, can only flip $0 \leftrightarrow 1$, while a qubit can drift anywhere on its Bloch sphere. Classical computers handle rare bit flips by copying bits redundantly, but the no-cloning theorem forbids this for qubits. QEC instead spreads one logical qubit's information across many entangled physical qubits, so that local errors can be detected and undone without ever reading the logical state itself.

how to implement difficult quantum algorithms, e.g., Shor's [2], Grover's [3]. They highlight the true advancement of quantum processing units (QPUs) compared to classical computers: Shor's algorithm factorises an n-bit integer in polynomial time $O(n^3)$, whereas the best known classical algorithm scales exponentially in n. Grover's algorithm finds a marked item in an unstructured set of size N in $O(\sqrt{N})$ queries instead of the classical $O(N)$. Concretely, Rivest–Shamir–Adleman (RSA) — the public-key cryptosystem that protects most HTTPS traffic and banking transactions — bases its security on the fact that factorising the product of two large primes is intractable classically. Breaking its 2048-bit standard would take to the fastest supercomputers an amount of time of the order of billions of years, whereas a fault-tolerant QPU is estimated to solve the same problem in a few hours [4].

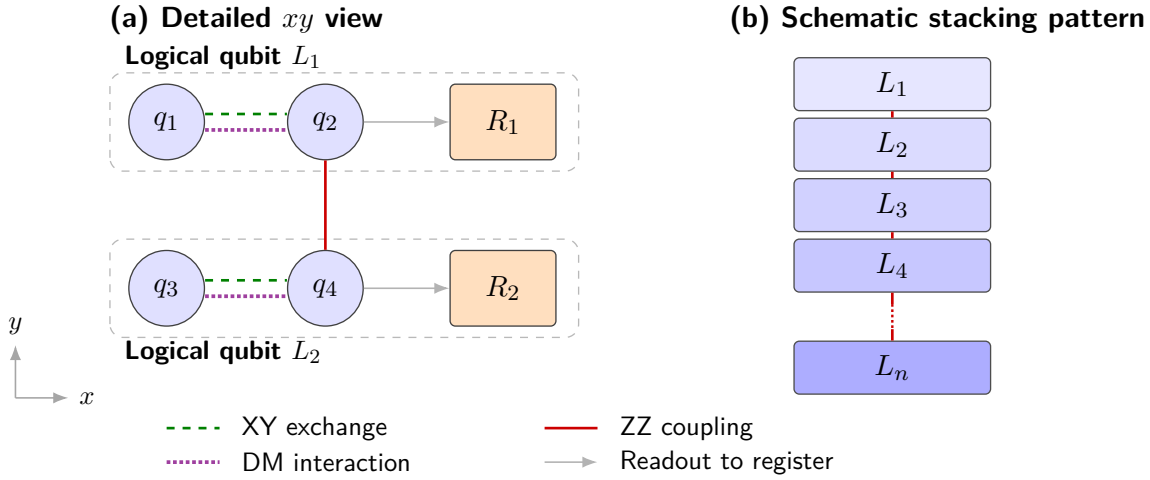


Figure 2: Schematic illustration of a logical qubit array, adapted and extended from Ref. [5]. **(a)** Detailed xy view of two logical qubits L_1 and L_2, each encoded with two physical qubits in the DFS encoding $|0\rangle_L = |01\rangle$, $|1\rangle_L = |10\rangle$. The XY exchange (dashed green) and DM interaction (dotted purple) act intra-pair within each logical qubit; the ZZ coupling (solid red) is drawn as an inter-pair coupling between q_2 and q_4 in the same plane. A readout register (R_1, R_2) is appended at the end of each logical qubit. **(b)** N qubits schematic stacking pattern: the same logical-qubit motif is tiled along a chain $L_1 \to L_2 \to L_3 \to L_4 \to \cdots \to L_n$, with a ZZ coupling between every pair of successive logical qubits. The readout registers represent the ancilla or the measurement block of the generic QEC processor of Fig. 1 [1], [6], and are not part of the decoherence-free-subspace scheme of Ref. [5].

Residing at the very bottom of the physical qubit layer are imperfect physical qubits, which have been explored in a variety of different experimental systems: superconducting qubits [7]–[10], trapped-ion [11]–[13], solid-state spin [14], [15], nuclear spin [16], [17], non linear photonic [18], and neutral atom qubits [19], which are just a few examples of more mature qubit systems demonstrating multi qubit operations. Coherence times in these systems are relatively varied[2]. For the purposes of quantum computing, it is important to normalize coherence to the gate (control) lengths possible for the system. For example, for superconducting qubits, with T_1 in the $\sim$ 100–300 µs range and gate lengths of $\sim$ 10–50 ns for single-qubit gates and $\sim$ 30–100 ns for two-qubit

[2]The coherence time of a qubit is the duration over which it preserves its quantum information before environmental noise destroys it. Past that point, the qubit loses its quantum state and must be re-initialized (DiVincenzo criterion 2 Sec. 1.1.5) for further computation.

gates, state-of-the-art devices reach on the order of 10^3–10^4 operations per coherence time, depending on whether one counts single or two-qubit gates [20], [21]. Inputs and outputs to the layer of physical qubits are controls and readouts, respectively. For superconducting qubits, controls can involve a full suite of microwave electronics and pulse shaping, to realize specific qubit rotations[3] and two-qubit controlled operations. The noise on these controls is filtered and attenuated so that the noise at the qubit is negligible. It is also important for readout of superconducting qubits to be boosted through stages of amplification, the first of which is quantum limited[4] [23]. Analog readout signals are then digitally processed either on classical computers or in customized field programmable gate arrays (FPGAs) for fast processing.

1.1.1 Physical and logical qubits

A physical qubit is the two-level quantum system actually realised in hardware - a transmon, a trapped ion, a spin, etc. Their associated imperfections, such as: decoherence, gate noise, and leakage, are the subject of Secs. 1.1.2 and 1.1.3. Since no foreseeable physical platform will deliver fidelities[5] high enough to run long quantum algorithms directly, the field has converged on a layer of abstraction sitting on top of the physical qubits.

A logical qubit is the abstract, error-corrected two-level system that this algorithm layer is written for. Its computational basis states $|0\rangle_L$ and $|1\rangle_L$ are not stored in any single physical device but encoded non-locally in the joint quantum state of several physical qubits, so that no individual physical qubit carries the logical information by itself. The machinery that turns this encoding into an actively error-corrected resource is the subject of the next subsections.

The simplest illustration of the encoding principle is sketched in Fig. 2, adapted from Ref. [5]. Two physical qubits are grouped to form one logical qubit through the decoherence-free subspace (DFS) encoding:

$$|0\rangle_L = |01\rangle\,, \qquad |1\rangle_L = |10\rangle\,. \tag{1}$$

Both logical basis states sit in the single-excitation subspace and share the same energy with respect to any collective dephasing noise that couples identically to the two physical qubits: such correlated noise acts as a global phase on the logical subspace and leaves the encoded information untouched. Within each logical qubit, single-qubit logical gates are implemented by the intra-pair XY exchange interaction (dashed green line in the figure) combined with a Dzyaloshinskii–Moriya (DM) interaction (dotted purple line), while a residual ZZ coupling (see Annex 5.1) between physical qubits belonging to different logical qubits (solid red line) mediates two-logical-qubit entangling gates.

This minimal 2:1 encoding only protects against one specific noise channel and does not implement a full error-correcting code. But it captures the essential idea common to every fault-tolerant scheme the logical information lives in a protected subspace of a larger physical Hilbert space, and

[3] Each qubit operation is equivalent to a rotation on the Bloch Sphere. More on the Bloch sphere on Chapter 2 of Ref. [22]

[4] Quantum-limited means the amplifier adds the minimum noise allowed by quantum mechanics (half a photon per mode). This is required because the readout signal only contains a few photons.

[5] Gate fidelity is a measure of how accurately a quantum gate performs its intended operation compared to the ideal, perfect version of that gate.

the logical operations are engineered interactions between physical qubits that preserve this subspace.

The DFS scheme in Fig. 2 protects the encoded information against collective dephasing, but it does not allow the active detection of errors: it contains only data qubits[6]. For a logical qubit to be usable inside the full fault-tolerant architecture of Fig. 1, it must in addition be able to feed the QEC processor with information about the errors it undergoes, without itself being measured, since this would collapse the encoded state. Following the surface-code paradigm[7] and the layered processor description of Ref. [1], this is achieved by appending to each logical qubit a dedicated readout register[8] (R_1, R_2 in Fig. 2): a small ancilla qubit that is briefly entangled with the data qubits, then projectively measured, returning a single classical bit (the syndrome[9]) without collapsing the logical state. The syndrome is then decoded in real time by the control layer of Fig. 1, which decides on and applies the appropriate correction back on the data qubits. The register is therefore an addition to the original diagram [5], not part of the DFS encoding itself, but it is what allows the logical qubit to plug into the QEC processor. Throughout the rest of this work, "register" refers to this ancilla measurement block attached to each logical qubit, distinct from the data qubits that hold the protected information.

Fig. 1 shows how this two-layer abstraction propagates up to the full processor architecture: a classical control layer drives the physical qubits and continuously runs error correction, while the algorithm sees only the error-corrected logical qubits sitting on top. The number of logical qubits effectively available at the top of this stack, rather than the raw physical qubits at the bottom, is what ultimately determines whether a given device can run a useful application.

1.1.2 Gates and gate noise

Since qubits are always subject to various forms of noise, and physical gate operations cannot be made perfect, it is widely recognized that large-scale, useful quantum computation is impossible without active error correction. This means that the $2^{1000} \sim 10^{300}$ continuously evolving quantum

[6] In quantum error correction, the qubits that actually carry the encoded logical state are called data qubits, in order to distinguish them from the ancilla qubits introduced solely for syndrome extraction (see Sec. 1.1.3 and Ref. [6]). In the DFS encoding of Fig. 2, the four physical qubits q_1, q_2, q_3, q_4 are all data qubits.

[7] The surface code is a two-dimensional stabilizer code in which the qubits are placed on a square lattice and the stabilizers are local four-body Pauli operators, alternating $XXXX$ and $ZZZZ$ on adjacent plaquettes of the lattice. It is currently the dominant QEC code for superconducting hardware because of its comparatively high error threshold (of the order of 1% per operation), its strictly planar layout, and the fact that it only requires nearest-neighbour two-qubit interactions [6].

[8] In the standard surface-code literature [6] this ancilla / measurement block is called a measure qubit (split into measure-Z and measure-X qubits) or, more generically, an ancilla qubit. Throughout this thesis we use the term register for this block, by analogy with the classical-CPU register that holds intermediate values before they are written back. It should not be confused with the textbook notion of a quantum register, which refers to a multi-qubit system as a whole and which is the standard meaning of the word in the wider quantum-computing literature.

[9] The syndrome is the bit string formed by the outcomes of stabilizer measurements, a stabilizer being a multi-qubit Pauli operator chosen so that every valid encoded state is its $+1$ eigenvector. As long as no error has occurred, every stabilizer measurement returns $+1$ and the syndrome is null. A flipped bit in the syndrome signals that an error anticommuting with the corresponding stabilizer has occurred, without ever revealing the logical state itself. A classical decoder then uses the syndrome to infer the most likely error and apply the matching correction on the data qubits [6], [24].

amplitudes of the wavefunction describing the full state of the computer must closely follow the desired evolution imposed by the quantum algorithm. Random drift of these amplitudes—caused by decoherence, gate imperfections, unwanted interactions, leakage, and other mechanisms—must be efficiently suppressed.

1.1.3 Quantum error correction

Building on the QEC principle introduced in Sec. 1.1.1, the logical qubit's information is distributed across the correlations between many physical qubits rather than being stored in any individual one. Errors on individual physical qubits can then be detected and corrected by measuring these correlations, called stabilizer measurements, without ever directly measuring the logical qubit itself and collapsing its state.

It is not obvious, even in principle, that error correction can be performed on an analogue machine whose state is described by so many continuous variables. Nevertheless, it is generally accepted [25] that fault-tolerant quantum computation [24], [26]–[28] using the techniques of error-correcting codes [29], [30] and concatenation (recursive encoding) provide a theoretical solution to this problem. Through active intervention, errors arising from noise and gate imperfections can be detected and corrected during computation without disturbing the encoded logical information.

A particularly prominent modern approach is the surface code [6], which achieves high error thresholds and requires only nearest-neighbour (NN) interactions, making it experimentally attractive. However, fault-tolerant operation comes at a steep overhead: realising a single logical qubit with sufficiently low error rates currently requires on the order of 100–1,000 physical qubits. That means to run a useful fault-tolerant algorithm requiring 1,000 logical qubits, one might need on the order of millions of physical qubits, especially for low coherence time qubits such as superconducting qubits. Recent research has improved decoherence from the order of microseconds to milliseconds [31] and this has contributed in reducing the number of physical qubits required for one logical qubit. Historically, the amount of physical qubits on a single QPU was the main metric to assess performances of recent breakthrough in the sector, but could shift to logical qubits as many companies now include logical qubits in their roadmaps, and are able to fabricate QPU's containing hundreds to thousands physical qubits (more on major actors in quantum computing and QPU's latest state of the art developments in Sec. 1.2).

1.1.4 Threshold theorem

The threshold theorem [32]–[34] states that, once the error per qubit per gate falls below a certain threshold value, arbitrarily long quantum computations become feasible at the cost of a polynomial overhead in the number of qubits. For the surface code, numerical and analytical studies place this threshold at roughly 10^{-2}–10^{-3} (i.e. 0.1–1%), a substantially less demanding target than earlier concatenated-code estimates of 10^{-6}–10^{-4}.

Theorists therefore claim that the problem of quantum error correction has been solved in principle, so that experimentalists and engineers need only to identify suitable qubit platforms and approach the accuracy required by the threshold theorem:

> "As it turns out, it is possible to digitize quantum computations arbitrarily accurately,

using relatively limited resources, by applying quantum error-correction strategies developed for this purpose." [35]

All hopes for scalable quantum computing rest ultimately on the threshold theorem.

Recent years have seen significant experimental progress. Demonstrations of quantum computational advantage on specific tasks have been reported by several groups [36]–[38], and small instances of error-corrected logical qubits have been realised in the laboratory [39], [40]. Nevertheless, operating a fault-tolerant quantum computer at a scale relevant to practical problems, such as breaking public-key cryptography[10] [2], remains a huge engineering challenge that has not yet been met.

1.1.5 DiVincenzo's criteria for constructing a quantum computer

In 2000, DiVincenzo introduced five criteria for what any physical platform must provide to build a general-purpose quantum computer [41]:

1. **A scalable physical system with well-characterised qubits.** The platform must offer identifiable two-level systems whose Hamiltonian is known, and the fabrication process must in principle allow scaling to many qubits.

2. **Reliable initialisation.** The qubits must be preparable on demand in a simple reference state such as $|00\ldots0\rangle$.

3. **Long coherence times relative to the gate time.** As a rule of thumb, the ratio T_1/T_{gate} should be at least 10^4, so that many operations can be performed before decoherence destroys the quantum information.

4. **A universal set of quantum gates.** Typically a set of single-qubit rotations combined with one entangling two-qubit gate such as CNOT or CZ, from which every unitary can be synthesised.

5. **Qubit-specific measurement.** It must be possible to read out the state of a chosen qubit at the end of the computation.

These five conditions remain necessary for any practical quantum computing platform, and every serious QPU manufacturer is still evaluated against them. However, following the recent progress on fault-tolerant quantum computing discussed in Sec. 1.1.3, the modern bar is significantly higher: real platforms must also achieve gate and measurement errors below the fault-tolerance threshold (Sec. 1.1.4), encode information in logical rather than physical qubits, provide sufficient qubit-to-qubit connectivity to limit SWAP overhead[11], and support fast mid-circuit measurement with

[10] As discussed at the beginning of this section, most secure internet traffic (HTTPS, banking, digital signatures) relies on the RSA public-key cryptosystem, whose security rests on integer factorisation. Shor's algorithm reduces this task to polynomial time, but a useful run on a 2048-bit key is estimated to require of the order of a few million physical qubits organised into thousands of fault-tolerant logical qubits [4], setting the magnitude of the engineering challenge referred to here.

[11] The two-qubit SWAP gate exchanges the states of two qubits: the state of the first becomes the state of the second, and vice-versa. Since two-qubit gates can only be applied on physically adjacent qubits, when the algorithm calls for a gate between two qubits that are not neighbours on the chip, SWAP gates have to be inserted to route them next to each other before the intended gate can be applied. Providing higher qubit-to-qubit connectivity is therefore part of the challenge of designing a reliable QPU: fewer SWAP insertions means fewer overhead operations, and therefore more **useful** gate operations fitting inside a single coherence time.

feedforward[12] for real-time syndrome extraction. DiVincenzo's criteria are therefore necessary but no longer sufficient conditions for a useful quantum processor.

1.1.6 Couplers

Independently of the qubit type, modern quantum processing units (QPUs) rely on couplers, a dedicated circuit element that mediate and control the interaction between neighboring qubits. Without such an element, two neighboring qubits are subject to various types of noise and can either exhibit unwanted always-on interactions through stray electromagnetic fields (NN interaction), or remain effectively decoupled when placed too far apart; both cases are detrimental to quantum gate operation. Depending on the architecture, a coupler can be designed as: (i) a switch, enabling programmable and toggleable interactions between two neighboring qubits by tuning its effective coupling strength to zero at idle and to a finite value during gate operations (Sec. 1.1.8); (ii) a fixed coupler, providing a constant, geometry-defined interaction strength between two neighboring qubits (Sec. 1.1.7); or (iii) a passive mediating element, an auxiliary circuit — typically a resonator or ancilla qubit — placed between the two qubits to channel their interaction through a shared electromagnetic mode, physically separating them while still enabling a controlled effective coupling whose strength is set by the mediator's properties rather than by direct qubit-qubit proximity (Sec. 1.1.9).

1.1.7 Direct couplers

The most straightforward coupling approach is to connect two qubits directly through a shared capacitive element C_{12}, as illustrated in Figure 3, forming what is known as a direct coupler.

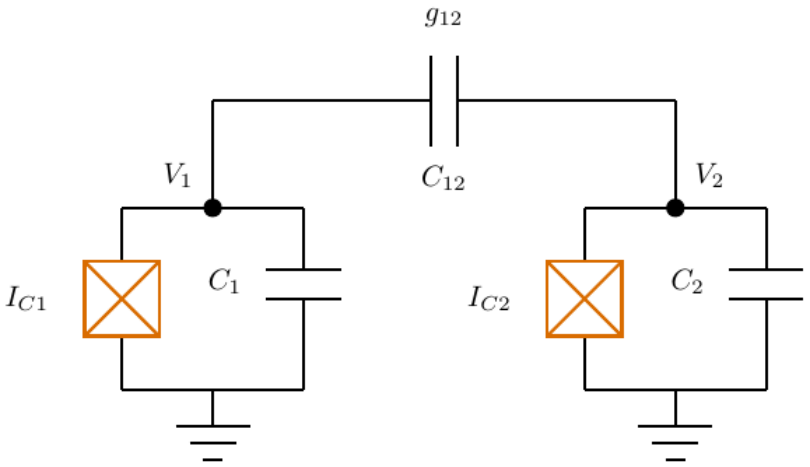


Figure 3: Circuit illustration of capacitive coupling adapted from Ref. [20]. A pair of transmons is connected through an intermediate capacitance C_{12} , and each transmon is represented as a Josephson-Junction of current I_{Ci} and capacitance C_i ; $i = 1, 2$. V_i, $i = 1, 2$ is the voltage at the node between the transmon and the intermediate capacitance C_{12} .

[12] Feedforward means using the outcome of a mid-circuit measurement to decide, in real time, which quantum gate to apply next on qubits that are still coherent. It is not to be confused with the ordinary feedback loop of the QEC processor: feedforward has to complete the classical decoding and response on a nanosecond-to-microsecond timescale — fast enough that the qubits acted upon have not yet decohered — as explained in Sec. 1.1.3.

The full Hamiltonian[13] of the two-qubit system reads

$$\hat{H} = \hat{H}_1 + \hat{H}_2 + \hat{H}_{\text{int}}, \tag{2}$$

where each single-transmon term, derived in Eqs. (17)–(20) of Ref. [20] from the standard quantization[14] of an anharmonic LC[15] oscillator, takes the form

$$\hat{H}_i = \hbar\omega_i \hat{a}_i^\dagger \hat{a}_i + \frac{\hbar\alpha_i}{2} \hat{a}_i^\dagger \hat{a}_i^\dagger \hat{a}_i \hat{a}_i, \qquad i = 1, 2. \tag{3}$$

In this expression, $\hat{a}_i$ and $\hat{a}_i^\dagger$ are respectively the bosonic annihilation and creation operators of transmon i, satisfying $[\hat{a}_i, \hat{a}_i^\dagger] = 1$; where the number operator $\hat{a}_i^\dagger \hat{a}_i$ counts the number of excitations on the transmon. The constant $\hbar$ is the reduced Planck constant, ω_i is the transmon angular frequency (i.e., the energy spacing between the ground state $|0\rangle_i$ and the first excited state $|1\rangle_i$, in units of $\hbar$), and $\alpha_i < 0$ is the anharmonicity (the difference in energy spacing between the $|1\rangle_i \to |2\rangle_i$ transition and the $|0\rangle_i \to |1\rangle_i$ transition). The negative anharmonicity is what allows the transmon to be addressed as an effective two-level system: excitations resonant with ω_i do not significantly populate the higher levels, but as seen in [42] real systems have imperfect anharmonicity that comes with its consequences. We focus here on the interaction term $\hat{H}_{\text{int}}$ that the coupler introduces, and on the resulting coupling constant g.

Treating the two transmon islands as nodes of a circuit and using the node fluxes[16] Φ_1, Φ_2 as generalized coordinates, the capacitive contribution to the Lagrangian is

$$\mathcal{L}_C = \tfrac{1}{2} C_1 \dot{\Phi}_1^2 + \tfrac{1}{2} C_2 \dot{\Phi}_2^2 + \tfrac{1}{2} C_g (\dot{\Phi}_1 - \dot{\Phi}_2)^2, \tag{4}$$

where C_1 and C_2 are the self-capacitances of the two transmons to ground, C_{12} is the shared capacitance of the coupler, $\dot{\Phi}_i$ denotes the time derivative of the node flux (i.e., the voltage at node i), and the cross term in $(\dot{\Phi}_1 - \dot{\Phi}_2)^2$ encodes the fact that the shared capacitance sees the voltage difference between the two transmons. This quadratic form is captured by the capacitance matrix

$$\mathbf{C} = \begin{pmatrix} C_1 + C_g & -C_g \\ -C_g & C_2 + C_g \end{pmatrix}, \tag{5}$$

whose diagonal elements $C_i + C_g$ represent the total capacitance seen by each node, and whose off-diagonal element $-C_g$ directly encodes the coupling. Introducing the conjugate node charges $Q_i = \partial\mathcal{L}/\partial\dot{\Phi}_i$ (each Q_i representing the total charge stored on transmon i), the kinetic part of the Hamiltonian becomes $H_C = \frac{1}{2}\boldsymbol{Q}^T \mathbf{C}^{-1} \boldsymbol{Q}$ where $\boldsymbol{Q} = (Q_1, Q_2)^T$, and the cross term of $\mathbf{C}^{-1}$ generates the interaction

[13] The Hamiltonian $\hat{H}$ of a quantum system is the operator representing its total energy. Its eigenvalues are the allowed energy levels of the system, and its action on the state vector $|\psi\rangle$ governs the time evolution through the Schrödinger equation $i\hbar\partial_t|\psi\rangle = \hat{H}|\psi\rangle$. Specifying the Hamiltonian is therefore equivalent to specifying the complete dynamics of the system.

[14] Quantization is the process of converting continuous values into a set of small, discrete values.

[15] An LC oscillator is a resonant circuit made of an inductor L and a capacitor C. Once quantized it behaves as a quantum harmonic oscillator with equidistant energy levels. Replacing the linear inductor by a Josephson junction turns it into the anharmonic oscillator that is used as a transmon qubit; see Sec. 2.1 for the full derivation.

[16] In circuit quantization, the node flux Φ_i at a circuit node is defined as the time integral of the voltage at that node; it plays the role of generalized position, while the node charge Q_i plays the role of generalized momentum [20].

$$H_{\text{int}} = \frac{C_g}{(C_1 + C_g)(C_2 + C_g) - C_g^2} Q_1 Q_2. \tag{6}$$

Writing the node charges in terms of ladder operators, $\hat{Q}_i = i\, Q_{\text{ZPF},i}\,(\hat{a}_i^\dagger - \hat{a}_i)$ with the zero-point charge fluctuation[17] $Q_{\text{ZPF},i} = \sqrt{\hbar\omega_i C_i/2}$, the interaction term becomes a product of four cross terms in $\hat{a}_i^\dagger$, $\hat{a}_i$. Under the rotating-wave approximation[18], which is valid whenever $\omega_1 + \omega_2 \gg g$ (a condition satisfied by several orders of magnitude for typical transmon parameters, $\omega_i \sim 2\pi \times 5$ GHz and $g \sim 2\pi \times 10$ MHz), the counter-rotating terms can be discarded and the interaction Hamiltonian takes its final form

$$\hat{H}_{\text{int}} = \hbar g \left(\hat{a}_1^\dagger \hat{a}_2 + \hat{a}_1 \hat{a}_2^\dagger\right), \tag{7}$$

where the operators $\hat{a}_1^\dagger \hat{a}_2$ and $\hat{a}_1 \hat{a}_2^\dagger$ describe respectively the transfer of one excitation from transmon 2 to transmon 1 and the reverse, and g_{12} is the coupling constant

$$g_{12} = \frac{1}{2} \frac{C_g}{\sqrt{(C_1 + C_g)(C_2 + C_g)}} \sqrt{\omega_1 \omega_2}. \tag{8}$$

The coupling strength g has units of angular frequency, is set at fabrication by the geometric ratio $C_g/\sqrt{(C_1 + C_g)(C_2 + C_g)}$, and scales with the geometric mean of the two qubit frequencies. Combining Eqs. (3) and (7), the full Hamiltonian of two capacitively coupled transmons is therefore

$$\hat{H} = \sum_{i=1}^{2} \left[\hbar\omega_i\, \hat{a}_i^\dagger \hat{a}_i + \frac{\hbar\alpha_i}{2}\, \hat{a}_i^\dagger \hat{a}_i^\dagger \hat{a}_i \hat{a}_i\right] + \hbar g \left(\hat{a}_1^\dagger \hat{a}_2 + \hat{a}_1 \hat{a}_2^\dagger\right). \tag{9}$$

Once projected onto the two-level subspace of each transmon, this transverse exchange interaction generates iSWAP-type two-qubit gates, and in the dispersive regime $|\omega_1 - \omega_2| \gg g$ it gives rise to the residual ZZ coupling responsible for the always-on nearest-neighbor interactions discussed at the beginning of Sec. 1.1.6 [43].

1.1.8 Indirect couplers

The indirect coupling scheme of Ref. [44] suppresses the always-on direct interaction by physically separating the two qubits and mediating their interaction through a shared coupler. The coupler is modelled as a lumped-element LC resonator of frequency $\omega_r = 1/\sqrt{LC}$, capacitively connected to each qubit, as illustrated in Fig. 4. The two qubits, with bare frequencies ω_1 and ω_2, are not connected to each other ($g_{12} = 0$): any interaction must transit through the coupler.

[17] The zero-point fluctuation $Q_{\text{ZPF},i} = \sqrt{\hbar\omega_i C_i/2}$ quantifies the rms charge fluctuations of the transmon in its ground state, arising from the Heisenberg uncertainty between charge and flux.

[18] The rotating-wave approximation (RWA) consists in discarding the fast-oscillating counter-rotating terms ($\hat{a}_1^\dagger \hat{a}_2^\dagger$ and $\hat{a}_1 \hat{a}_2$, which oscillate at the sum frequency $\omega_1 + \omega_2$ in the interaction picture) while keeping the slowly-varying resonant terms $\hat{a}_1^\dagger \hat{a}_2 + \hat{a}_1 \hat{a}_2^\dagger$ (which oscillate at the much smaller detuning $|\omega_1 - \omega_2|$).

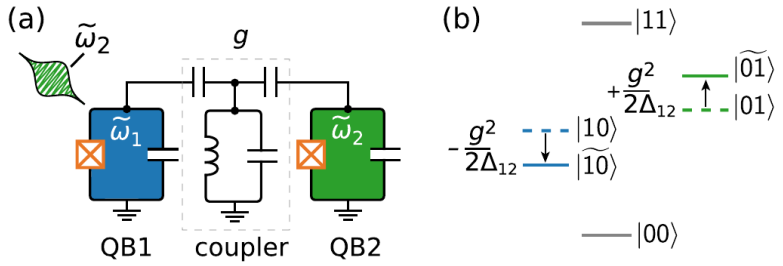


Figure 4: Indirect coupling of two fixed-frequency transmons through a lumped-element LC resonator (the coupler), adapted from Ref. [20]. **(a)** Schematic circuit: each transmon (QB1 with frequency $\tilde{\omega}_1$, QB2 with frequency $\tilde{\omega}_2$) is capacitively coupled to the coupler resonator, yielding an effective qubit-qubit coupling coefficient g given by Eq. (12). **(b)** Schematic level diagram of the always-on coupling: the bare two-qubit states $|10\rangle$ and $|01\rangle$ are dressed by the exchange interaction into $|\widetilde{10}\rangle$ and $|\widetilde{01}\rangle$ shifted by $\pm g^2/(2\Delta_{12})$ with $\Delta_{12} = \omega_1 - \omega_2$, while $|00\rangle$ and $|11\rangle$ are unaffected.

The bare three-mode Hamiltonian is the sum of two Jaynes-Cummings contributions

$$\hat{H} = \tfrac{\hbar\omega_1}{2}\,\hat{\sigma}_1^z \;+\; \tfrac{\hbar\omega_2}{2}\,\hat{\sigma}_2^z \;+\; \hbar\omega_r\,\hat{a}^\dagger\hat{a} \;+\; \sum_{j=1,2} \hbar g_j \left(\hat{\sigma}_j^+\hat{a} \;+\; \hat{\sigma}_j^-\,\hat{a}^\dagger\right), \tag{10}$$

where $\hat{a}$, $\hat{a}^\dagger$ are the bosonic ladder operators of the coupler mode and g_j is the qubit-coupler capacitive coupling strength of qubit j.

We assume the system is operated in the dispersive regime, with both qubits strongly detuned from the coupler, $|\Delta_j| = |\omega_j - \omega_r| \gg g_j$. In this limit, real photon exchange between the qubits and the coupler is energetically forbidden, and the coupler only mediates virtual-photon processes. The bus stays in its ground state at all times and a Schrieffer–Wolff transformation eliminates it to second order in g_j/Δ_j, yielding an effective two-qubit Hamiltonian of the form [20]

$$\hat{\tilde{H}} = \tfrac{\hbar\tilde{\omega}_1}{2}\,\hat{\sigma}_1^z \;+\; \tfrac{\hbar\tilde{\omega}_2}{2}\,\hat{\sigma}_2^z \;+\; \hbar g \left(\hat{\sigma}_1^+\hat{\sigma}_2^- \;+\; \hat{\sigma}_1^-\hat{\sigma}_2^+\right), \tag{11}$$

where $\tilde{\omega}_j = \omega_j + g_j^2/\Delta_j$ are the Lamb-shifted qubit frequencies (which absorb the qubit-coupler virtual exchange), and g is the resulting overall qubit-qubit coupling coefficient

$$g \;=\; \frac{g_1 g_2}{2}\left(\frac{1}{\Delta_1} + \frac{1}{\Delta_2}\right). \tag{12}$$

This effective coupling g is the single number that appears in Fig. 4(a): once the coupler has been integrated out, the two qubits behave as if they were directly capacitively coupled by a single strength g, even though the underlying coupling is mediated by virtual photons in the LC resonator. The qubit-qubit detuning is $\Delta_{12} = \omega_1 - \omega_2$. Fig. 4(b) shows the resulting level structure: the always-on coupling hybridises the bare states $|10\rangle$ and $|01\rangle$ into dressed states $|\widetilde{10}\rangle$ and $|\widetilde{01}\rangle$, shifted by $\pm g^2/(2\Delta_{12})$ relative to their bare positions, while $|00\rangle$ and $|11\rangle$ remain unaffected because they do not couple resonantly through the exchange interaction.

A central feature of this scheme is that the hardware coupling g is fixed at fabrication and is always on. The effective qubit-qubit interaction cannot be switched off *in situ* by modifying g itself, but only by detuning the two qubits from each other through their individual flux biases, since the exchange term in Eq. (11) becomes off-resonant whenever $|\Delta_{12}| \gg g$. The main practical benefit

of the bus is geometric: the two qubits can be placed centimetres apart on the chip, eliminating direct capacitive crosstalk, while the same resonator simultaneously serves as a dispersive readout element. The impossibility of dynamically switching g off motivates the introduction of the frequency-tunable coupler discussed in the next section.

1.1.9 Frequency-tunable couplers

As mentioned in Section 1.1.6, improving two-qubit gate fidelity is a high priority to realize large-scale quantum processors [45]. Apart from decoherence, the main source of gate error is non-ideal interactions, which include parasitic coupling, leakage to non-computational states, and control crosstalk. One important example of parasitic coupling is the next-nearest-neighbor (NNN) coupling, a phenomenon commonly seen in many qubit platforms, including Rydberg atoms, trapped ions, semiconductor spin qubits, and superconducting qubits [45]. The NNN coupling is a residual, always-on interaction between two qubits that are meant to be unconnected: that is, two qubits which sit two lattice steps apart and are only supposed to interact through an intermediate coupler. It is generally considered spurious and introduces unwanted entanglement between physically separated qubits.

A natural way to suppress these unwanted interactions is to insert a tunable coupling element, a coupler, between each pair of qubits, so that the actual qubit-coupler exchange, the nearest-neighbor (NN) coupling, can be turned on only during gate operation. The NN coupling is generally stronger than the NNN one, $g_j > g_{12} > 0$, $j = 1,2$ with g_j the qubit-coupler coupling and g_{12} the direct NNN qubit-qubit coupling. The frequency-tunable coupler scheme of Ref. [45] goes one step further: it turns the always-on NNN coupling, traditionally seen as a parasitic effect, into a useful resource that enables a perfect OFF state via destructive interference with the indirect channel mediated by the coupler. We now sketch the derivation of the effective two-qubit Hamiltonian behind this idea. The frequency-tunable coupler scheme of Ref. [45] introduces a third qubit used as a coupler between the two target qubits in order to make their effective coupling controllable *in situ*. We consider a generic system consisting of a chain of three modes with exchange coupling between nearest and next-nearest neighbors, as outlined in Fig. 5(a).

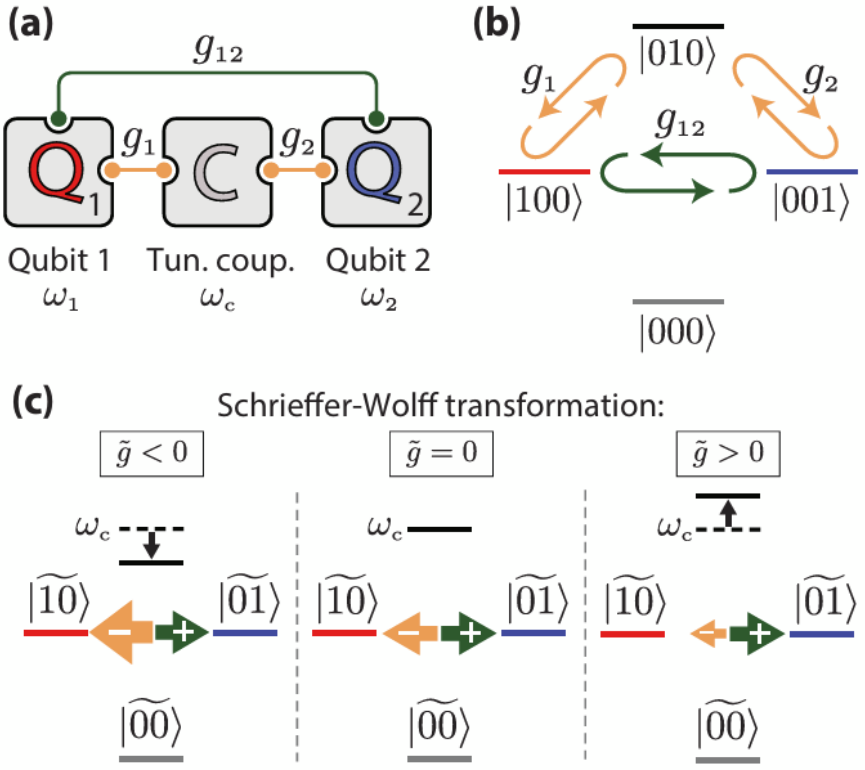


Figure 5: **(a)** Sketch of the generic three-body system in a chain geometry: two qubits (Q1, Q2) with bare frequencies ω_1, ω_2 are each coupled to an intermediate tunable coupler of frequency ω_c via the nearest-neighbor (NN) qubit-coupler coupling strengths g_1, g_2, and are additionally coupled directly to each other by the next-nearest-neighbor (NNN) coupling g_{12}, typically parasitic in fixed-frequency architectures. **(b)** Sketch of the three states of the system, labeled $|Q_1, c, Q_2\rangle$, in which a single excitation sits either on qubit 1, on the coupler, or on qubit 2. It shows the two competing interaction channels: the direct NNN coupling g_{12} that swaps the excitation between the two qubits, $|100\rangle \leftrightarrow |001\rangle$ (bottom arrow), without ever populating the coupler, and the indirect two-step path that briefly hops through the coupler, $|100\rangle \overset{g_1}{\longleftrightarrow} |010\rangle \overset{g_2}{\longleftrightarrow} |001\rangle$ (top arrows). **(c)** Effective qubit-qubit coupling $\tilde{g}$ as a function of the coupler frequency ω_c: the direct term $g_{12} > 0$ and the indirect term (negative in the dispersive regime) have opposite signs, so their sum $\tilde{g}(\omega_c)$ crosses zero at a well-defined coupler frequency ω_c^{off}. Tuning ω_c across this point continuously varies the effective coupling from negative, through zero (OFF condition), to positive, which is what enables high ON/OFF ratios in this coupler scheme. Adapted from [45].

The two qubits (ω_1 and ω_2) each couple to a center tunable coupler (ω_c) with a coupling strength g_j ($j = 1, 2$), as well as to each other with a coupling strength g_{12}. Without loss of generality, we begin our analysis with a two-level Hamiltonian

$$\hat{H} = \sum_{j=1,2} \tfrac{1}{2}\omega_j \hat{\sigma}_j^z + \tfrac{1}{2}\omega_c \hat{\sigma}_c^z + \sum_{j=1,2} g_j \left(\hat{\sigma}_j^+ \hat{\sigma}_c^- + \hat{\sigma}_j^- \hat{\sigma}_c^+\right) + g_{12}\left(\hat{\sigma}_1^+ \hat{\sigma}_2^- + \hat{\sigma}_2^- \hat{\sigma}_1^+\right), \tag{13}$$

where $\hat{\sigma}_\lambda^z$, $\hat{\sigma}_\lambda^+$ and $\hat{\sigma}_\lambda^-$ ($\lambda = 1, 2, c$) are, respectively, the Pauli-Z, raising and lowering operators defined in the eigenbasis of the corresponding mode.

We assume that both qubits are negatively detuned from the coupler, $\Delta_j \equiv \omega_j - \omega_c < 0$, and that the coupling is dispersive, $g_j \ll |\Delta_j|$ $(j = 1,2)$. The physical mechanism behind the switch is read directly on the one-excitation level diagram of Fig. 5(b), where a single excitation can travel from Q_1 to Q_2 along *two competing channels*. The *direct* channel $|100\rangle \leftrightarrow |001\rangle$ (bottom arrow) is mediated by the NNN coupling g_{12} and transfers the excitation without ever populating the coupler; its amplitude is $\propto +g_{12}$. The *indirect* channel $|100\rangle \overset{g_1}{\longleftrightarrow} |010\rangle \overset{g_2}{\longleftrightarrow} |001\rangle$ (top arrows) is a two-step virtual process that briefly visits the coupler state $|010\rangle$ before returning to the qubit subspace; at second order in perturbation, its amplitude is $\propto -g_1 g_2/|\Delta|$. Crucially, the two amplitudes carry *opposite signs*, because the qubits are detuned below the coupler ($\Delta_j < 0$). This sign mismatch is what opens the door to destructive interference between the two channels, and ultimately defines the OFF point of the switch. The indirect channel is sometimes called the virtual exchange interaction [46], and can be systematically integrated out by the Schrieffer–Wolff transformation (SWT) [47]. Assuming that the coupler mode remains in its ground state at all times, the SWT decouples the coupler from the system up to second order in g_j/Δ_j, and yields the effective two-qubit Hamiltonian

$$\hat{\tilde{H}} \;=\; \sum_{j=1,2} \tfrac{1}{2}\,\tilde{\omega}_j\,\hat{\sigma}_j^z \;+\; \tilde{g}(\omega_c)\left(\hat{\sigma}_1^+\hat{\sigma}_2^- \;+\; \hat{\sigma}_2^-\hat{\sigma}_1^+\right), \tag{14}$$

with Lamb-shifted qubit frequencies $\tilde{\omega}_j = \omega_j + g_j^2/\Delta_j$, and an effective qubit-qubit coupling strength

$$\tilde{g}(\omega_c) \;=\; g_{12} \;+\; \frac{g_1 g_2}{2}\left(\frac{1}{\Delta_1} \;+\; \frac{1}{\Delta_2}\right). \tag{15}$$

Equation (14) captures the operational role of the coupler: the two qubits behave as if they were directly coupled by a single strength $\tilde{g}$, while the coupler itself has been removed from the effective dynamics. Equation (15) shows that $\tilde{g}$ is the sum of two contributions of opposite sign. The first term, $g_{12} > 0$, is the direct NNN coupling, fixed at fabrication through the intermediate capacitance C_{12} between the qubits (see Fig. 6). The second term, $\frac{1}{2}\,g_1 g_2(1/\Delta_1 + 1/\Delta_2) < 0$, is the indirect virtual-exchange coupling through the coupler; it is negative because both detunings are negative, and its magnitude depends on ω_c through Δ_j. Both g_1 and g_2 may also depend implicitly on ω_c, so $\tilde{g}$ is in general a continuous function of the coupler frequency.

As illustrated in Fig. 5(c), $\tilde{g}(\omega_c)$ can be tuned positive when the coupler frequency is increased, or negative when it is decreased. Most importantly, since the tunability is continuous, there always exists a critical value ω_c^{off} at which the two contributions cancel and the qubits are effectively decoupled

$$\tilde{g}(\omega_c^{\text{off}}) \;=\; 0, \tag{16}$$

as long as it lies within the bandwidth of the coupler. The dispersive-limit condition is only an ideal requirement: in systems with considerably greater g_{12}, it is still possible to find such an ω_c^{off} in the weakly dispersive regime ($g_j < |\Delta_j|$).

The qubit-coupler coupling strengths g_j are set by the capacitive network connecting each qubit to the coupler transmon. Following the same circuit-quantization procedure as in Sec. 1.1.7, the result is [20]

$$g_j = \frac{1}{2} \frac{C_{g,j}}{\sqrt{\left(C_{\Sigma,j} + C_{g,j}\right)\left(C_{\Sigma,c} + C_{g,j}\right)}} \sqrt{\omega_j\, \omega_c}, \tag{17}$$

where $C_{g,j}$ is the qubit-coupler coupling capacitance, $C_{\Sigma,j}$ is the total capacitance of qubit j to ground, and $C_{\Sigma,c}$ is the total capacitance of the coupler island to ground. The NN couplings g_j are therefore fixed at fabrication by the geometric capacitive ratio, while the prefactor $\sqrt{\omega_j \omega_c}$ inherits the tunability of ω_c. Inserted into Eq. (15), this fully determines $\tilde{g}$ in terms of fabrication parameters and the single in-situ knob ω_c.

The frequency-tunable coupler scheme has three key advantages over a direct, always-on coupler [45]. First, it incorporates the always-on NNN coupling g_{12}, traditionally seen as a parasitic effect, directly into the switch: the destructive interference between g_{12} and the indirect virtual-exchange channel is what makes the perfect OFF state of Eq. (16) achievable. Second, two-qubit gate operations are performed with the coupler kept in the dispersive regime, which strongly suppresses leakage to the coupler's excited states and leaves the qubits themselves nearly unperturbed by the control signal. Third, the scheme is compatible with existing high-coherence hardware: the coupler is implemented as an additional transmon connected to each qubit via fixed capacitances, so no extra decoherence channel is introduced besides the coupler itself.

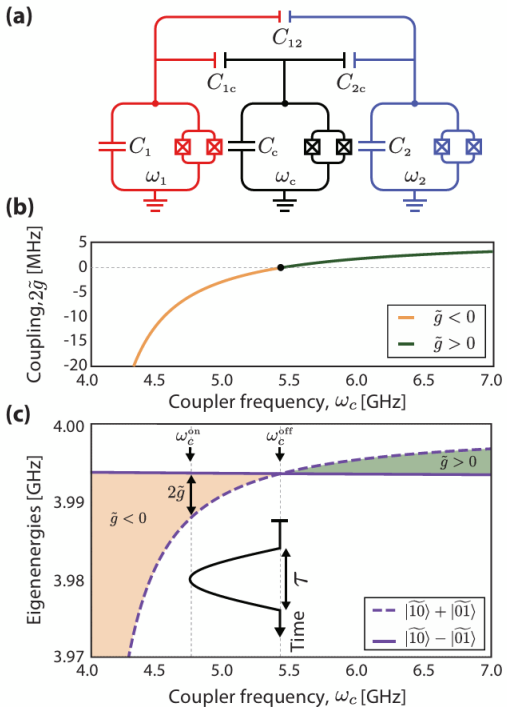


Figure 6: Circuit-level implementation of the frequency-tunable coupler scheme and its operating characteristic, adapted from Ref. [45]. **(a)** Lumped-element circuit: three flux-tunable transmons (each Josephson junction replaced by a SQUID loop, see Sec. 1.1.9) linked by a purely capacitive network. Qubit Q_1 (red, self-capacitance C_1, frequency ω_1) and Q_2 (blue, C_2, ω_2) are each coupled to the central coupler (black, C_c, ω_c) through the qubit-coupler capacitances C_{1c}, C_{2c}, and directly to each other through the NNN capacitance C_{12}. This capacitive network fixes the NN couplings g_j of Eq. (17) and the direct NNN qubit-qubit coupling g_{12} entering the effective Hamiltonian (14). **(b)** Effective qubit-qubit coupling $2\tilde{g}$ as a function of ω_c, obtained by inserting $g_j(\omega_c)$ and g_{12} into Eq. (15). The OFF frequency ω_c^{off} (black dot) marks the zero-crossing where the direct (positive) and virtual-exchange (negative) contributions cancel; to the left of ω_c^{off} (orange) $\tilde{g} < 0$, to the right (green) $\tilde{g} > 0$. **(c)** Eigenspectrum of the one-excitation manifold as a function of ω_c, at resonance $\omega_1 = \omega_2$. The two singly-excited eigenstates are the symmetric and anti-symmetric dressed combinations $|\widetilde{10}\rangle \pm |\widetilde{01}\rangle$ of the bare qubit states; their gap (shaded) equals $2|\tilde{g}(\omega_c)|$ and closes exactly at ω_c^{off}. The inset sketches the cosine pulse of duration τ that modulates ω_c from the idle value ω_c^{off} to the active value ω_c^{on}: choosing τ such that $\int_0^\tau 2\tilde{g}(t)\,dt = 1/2$ implements an iSWAP gate between Q_1 and Q_2.

At the hardware level, the *in-situ* tunability of ω_c is realised by replacing the single Josephson junction of the coupler transmon with a Superconducting Quantum Interference Device (SQUID): a small superconducting loop of two Josephson junctions in parallel, as seen in Fig. 6(a). An external magnetic flux threading the SQUID loop, applied through a dedicated on-chip flux line, modifies the effective Josephson energy of the coupler and therefore its plasma frequency ω_c. We can therefore tune the expression of g_{12} based on the resulting lumped circuit:

$$g_{12} \approx \frac{1}{2}\left[\frac{C_{12}}{\sqrt{C_{\Sigma,1}\,C_{\Sigma,2}}} + \frac{C_{g,1}\,C_{g,2}}{\sqrt{C_{\Sigma,1}\,C_{\Sigma,2}}\,C_{\Sigma,c}}\right]\sqrt{\omega_1\,\omega_2}, \tag{18}$$

where C_{12} is the direct island-to-island capacitance between the two qubits (top branch of Fig. 6(a)),

and $C_{g,1}$, $C_{g,2}$, $C_{\Sigma,c}$ are the qubit-coupler and coupler-to-ground capacitances already defined in Eq. (17). The first term of Eq. (18) is the direct capacitive path between the two qubit islands; the second is a residual coupling mediated by the intermediate coupler network, which survives even in the limit $C_{12} \to 0$. This is what makes the OFF condition (Eq. (16)) generic: g_{12} is essentially never zero in a realistic layout, so the destructive interference with the indirect channel can always be tuned in at a finite ω_c^{off}.

Inserting the capacitive expressions Eq. (17) and Eq. (18) into Eq. (15) turns the analytical picture of Fig. 5(c) into a concrete prediction for the device of Fig. 6(a). The resulting effective coupling $2\tilde{g}(\omega_c)$, plotted in Fig. 6(b) as the SQUID coupler is tuned in the [4, 7] GHz range, crosses zero at $\omega_c^{\text{off}} \approx 5.5$ GHz and reaches amplitudes of order MHz on either side — the concrete magnitude of the qubit-qubit interaction that can be switched on and off with this scheme.

Moreover, the OFF condition $\tilde{g}(\omega_c^{\text{off}}) = 0$ translates directly into a closing of the energy gap between the two singly-excited eigenstates of the system, $|\widetilde{10}\rangle \pm |\widetilde{01}\rangle$, whose splitting equals $2|\tilde{g}(\omega_c)|$ (Fig. 6(c), at resonance $\omega_1 = \omega_2$). This gives a straightforward experimental calibration of ω_c^{off}: one measures the Rabi oscillation frequency[19] between the two qubits while sweeping ω_c, and reads off the value at which the oscillation stops [45].

[19] When two quantum states are connected by an interaction that matches their energy difference, a system prepared in one of them does not stay there: it transfers all its population to the other state, then back, and so on. This periodic back-and-forth is called a Rabi oscillation. Its frequency is set by the strength of the coupling between the two states, $2|\tilde{g}(\omega_c)|$ here, and therefore vanishes when the coupling is switched off.

1.2 Major actors in quantum computing

The race to build a useful quantum computer is now a global industrial and academic effort. As of 2025, more than two dozen manufacturers commercially offer quantum processing units (QPUs), yet none of these devices meet the requirements for running large-scale commercial applications such as chemical simulation or cryptanalysis [48]. The technology is maturing rapidly, however, with a handful of leading players — most notably Google, IBM, and Microsoft — setting the pace of hardware development. These companies differ not only in qubit count but also in the fundamental physical realisation of their qubits, their error-correction strategies, and their software ecosystems.

1.2.1 IBM Quantum

IBM occupies a unique position in the field by combining hardware development with an open-access software ecosystem from the outset. In 2016, IBM placed the first quantum computer on the cloud, allowing anyone to run experiments remotely [49]. One year later the company released Qiskit, an open-source Python-based framework for programming quantum hardware that has since become the most widely used quantum software toolkit worldwide.

On the hardware side, IBM has followed an aggressive qubit-scaling roadmap (Figure 7). Key milestones include the 27-qubit *Falcon* (2018), the 65-qubit *Hummingbird* (2020), the 127-qubit *Eagle* (2021) — the first processor to exceed 100 qubits — the 433-qubit *Osprey* (2022), and the 1,121-qubit *Condor* (2023). Alongside raw qubit count, IBM has placed increasing emphasis on gate fidelity: the *Heron* processor achieved the lowest median two-qubit gate error rates in the company's history, with 57 of its 176 possible two-qubit couplings delivering fewer than one error per thousand operations.

The most recent generation is the *Nighthawk*, a 120-qubit chip arranged in a square lattice topology, announced at the IBM Quantum Developer Conference in 2025 [50]. Its square layout increases the number of couplers from Heron's 176 to 218, enabling circuit designs that are 30% more complex with fewer SWAP gates. IBM projects that the Nighthawk line will scale through successive revisions capable of executing 5,000, 7,500, 10,000, and ultimately 15,000 quantum gates.

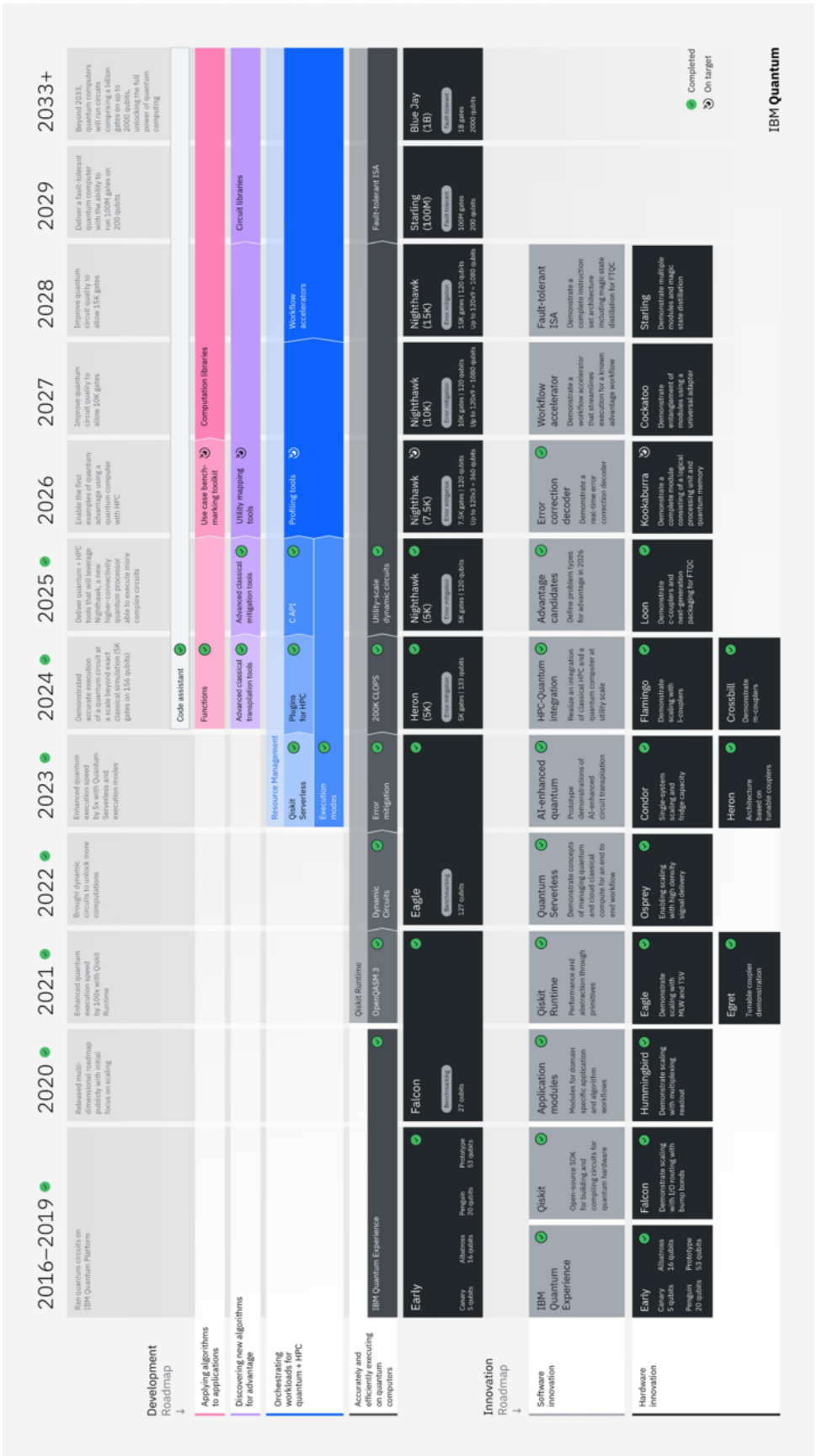


Figure 7: IBM Quantum Development and Innovation Roadmap (2025), illustrating the progression from near-term noisy devices towards large-scale fault-tolerant quantum computing. Source: IBM Quantum.

1.2.2 Google Quantum AI

Google's quantum computing programme has focused on superconducting qubits and on demonstrating hardware advantages over the best available classical systems. In 2018 the company unveiled *Bristlecone*, a 72-qubit superconducting processor used to probe the limits of quantum error rates at scale [51].

The landmark result came in 2019 with *Sycamore*, a 53-qubit device. Google claimed that Sycamore had achieved *quantum supremacy*: the ability of a quantum processor to complete a specific task — random circuit sampling — that the most powerful available supercomputer could not finish in a practical amount of time [36]. The Sycamore processor completed the sampling task in approximately 200 seconds, a result estimated to require thousands of years on a classical machine, though the precise classical hardness of this benchmark has been contested in subsequent work. More recently, Google reported results on the *Willow* processor, demonstrating quantum error correction operating below the surface code threshold for the first time [38]. This is a landmark milestone: it confirms the core prediction of the threshold theorem experimentally — that increasing the number of physical qubits encoding a single logical qubit actually reduces the logical error rate, provided physical errors are kept below threshold. This so-called *below-threshold* operation is a prerequisite for any practical fault-tolerant quantum computer.

1.2.3 Microsoft Azure Quantum

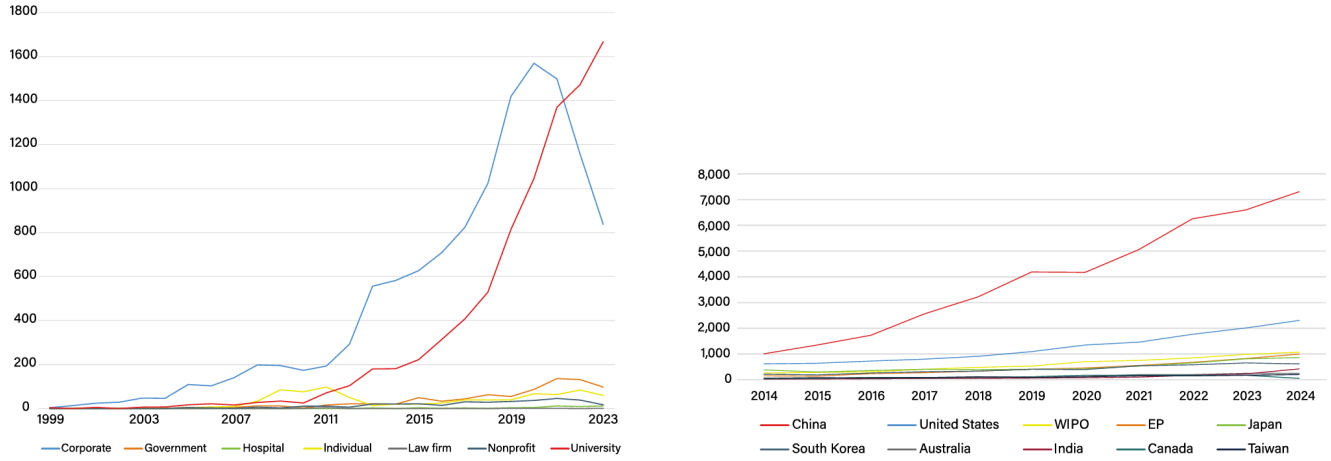
Microsoft has chosen a fundamentally different physical approach. Rather than pursuing superconducting or trapped-ion platforms, the company is developing *topological qubits*, which encode information in *Majorana zero modes* — exotic quasi-particles that arise at the boundaries of a new class of material called a *topoconductor* [52]. The central claimed advantage of this approach is inherent noise protection: because the logical information is encoded non-locally in the combined state of two spatially separated Majorana modes, local perturbations cannot corrupt it without accessing both ends simultaneously.

In February 2025, Microsoft unveiled *Majorana 1*, described as the world's first quantum processor built on a topological core architecture. The device is designed from the outset to scale to one million qubits on a single chip, a target that would be extraordinarily difficult to achieve with conventional superconducting architectures. While topological qubits remain at an earlier stage of experimental maturity than superconducting or trapped-ion alternatives, they represent one of the most compelling long-term routes to fault-tolerant quantum computing.

1.2.4 Broader landscape

Beyond these three leading players, a rich ecosystem of companies and academic groups is exploring diverse physical platforms including trapped ions (IonQ, Quantinuum), neutral atoms (Pasqal, QuEra), photonic systems (PsiQuantum, Xanadu), and quantum dots. A 2025 MIT report on the state of quantum computing noted that quantum technology patents have grown dramatically [48], with a rapid period of expansion from 2013 to 2019 driven almost entirely by corporations and universities, as shown in Fig. 8a. The corporate sector eventually reached its maximum of 1,570 patents in 2020, while universities continued strong growth to reach 1,668 patents in 2023, positioning them as drivers of quantum computing innovation. Government participation accelerated notably after 2019. The year 2023 also marked the first significant decline across most categories

except for universities, suggesting potential market adjustments. As shown in Fig. 8b, China alone accounts for 60% of quantum patents as of 2024, followed by the United States with around 19% and the World Patent Office with 9%. Together, these three entities controlled 88% of all quantum computing patents in 2024, indicating a highly concentrated intellectual property landscape in this technology sector. Standardisation efforts are underway at NIST and ISO, and, in 2024, NIST finalised the first post-quantum cryptographic standards in anticipation of the eventual cryptanalytic threat posed by fault-tolerant quantum machines [53].



(a) (b)

Figure 8: **(a)** Quantum technology patent families by origin, **(b)** quantum technology patents by country [48].

1.3 My thesis research project

The research presented in this thesis builds directly on Yohan Burignat's 2025 Master thesis [22], which proposed a fast, simulation-driven, pre-fabrication workflow for transmon-based superconducting quantum chips. Conventional design-build-test cycles are dominated by fabrication-induced deviations in qubit frequency, coupling strength and coherence times, which are then corrected only *a posteriori* through costly iterative prototyping. To circumvent this bottleneck, Burignat developed an *in silico* methodology aimed at translating a target Hamiltonian into a fabrication-ready layout while minimising reliance on physical prototypes.

The architecture studied is canonical: a single transmon qubit capacitively coupled to a $\lambda/4$ coplanar-waveguide resonator, itself capacitively coupled to a feedline for dispersive readout, as seen in Figure 9. Rather than fixing a precise application Hamiltonian, the flow is parameter-centric: the target quantities are the Josephson-to-charging energy ratio E_J/E_C, the qubit and resonator frequencies (with $f_r = 7$ GHz), the anharmonicity α,[20] and the relaxation time T_1. Those target parameters and the values obtained after simulation are recapitulated on Table 2.

[20] The anharmonicity is defined as $\alpha = (E_2 - E_1) - (E_1 - E_0)$, i.e. the difference (in units of $\hbar$) between the $|1\rangle \to |2\rangle$ and $|0\rangle \to |1\rangle$ transition frequencies. It vanishes for a harmonic oscillator and takes the perturbative value $\alpha \simeq -E_C/\hbar$ in the transmon regime. A finite $|\alpha|$ is what allows a microwave pulse tuned to ω_{01} to drive the qubit without exciting leakage to $|2\rangle$.

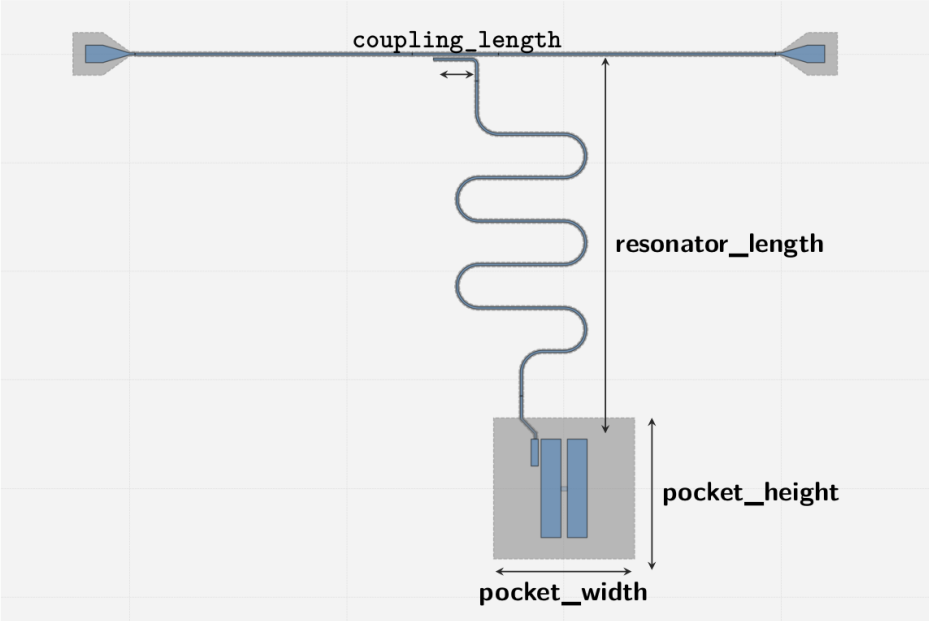


Figure 9: Parametric layout of Burignat's single-transmon chip [22]. A transmon pocket (two capacitor pads shunted by a Josephson junction) is capacitively coupled, through a meandered $\lambda/4$ coplanar-waveguide resonator, to a feedline used for dispersive readout. The four geometric quantities used as primary degrees of freedom in the parametric sensitivity study - `coupling_length`, `resonator_length`, `pocket_width` and `pocket_height` - are highlighted.

Parameter	**Symbol**	**Target**	**Obtained**
Josephson-to-charging energy ratio	E_J/E_C	> 50	62.8
Qubit frequency	$\omega_q/2\pi$	4.5–5.5 GHz	5.56 GHz
Anharmonicity	$\alpha/2\pi$	$-$(200–300) MHz	-293.4 MHz
Energy relaxation time	T_1	≈ 50 µs	32.2 µs

Table 2: Target vs. obtained values for Burignat's single-transmon reference chip. Targets are the design specifications listed in Sec. 3.3 of Ref. [22]. All obtained values come from the Ansys Q3D branch, Sec. 3.7. An additional target of 7 GHz is set up for the resonator, extracted from the HFSS branch and returned 6.59 GHz. Q3D cannot extract the frequency of the resonator, and Burignat set up an input of 7GHz (see Sec. 3.2.3)

Analytical formulas first seed the resonator and coupling geometry (*e.g.* for a $\lambda/4$ CPW resonator, the relation between the physical length l_r and the target resonance frequency f_r is

$$l_r = \frac{c_0}{4 f_r \sqrt{\varepsilon_{\text{eff}}}}, \tag{19}$$

where $c_0 = 3\times10^8$ m/s is the speed of light in vacuum, f_r is the target bare resonator frequency, and ε_{eff} is the effective permittivity of the resonator. This methodology is repeated among remaining target parameters and their respective relation with the geometry), the layout is then generated programmatically with the open-source *qiskit-metal* framework, and two complementary Ansys

branches are run in parallel on that geometry (Fig. 10).

The Quasi-static 3D (Q3D) branch treats the chip as an electrostatic problem: the Maxwell capacitance matrix is extracted from the layout and injected into a lumped-oscillator-model (LOM) quantization, which returns E_J/E_C, the qubit frequency, anharmonicity, dispersive shift χ and qubit–resonator coupling g. The junction itself is not simulated: $L_J = 10$ nH and $C_J = 2$ fF, together with the resonator frequency `freq_readout` $= 7.0$ GHz set by the analytical length l_r, enter the analysis as bare (undressed) subsystem parameters, which the extracted capacitance matrix then dresses into the effective quantities of Table 2. They are therefore inputs of the workflow, not results of it.

The High-Frequency Structure Simulator (HFSS) branch runs a full 3D eigenmode simulation with a linearised Josephson junction ($L_J \parallel C_J$) and is used only for modal validation: the fundamental linear mode frequencies (qubit at 6.18 GHz, resonator at 6.59 GHz, feedline at 6.95 GHz) and the associated electric field confirm that the intended mode structure is preserved and that the field energy is localised on the expected elements. An energy-participation-ratio (EPR) post-processing (pyEPR) that would upgrade these linear modes into non-linear Hamiltonian parameters (α, χ, g) is set up but not critically analysed, and none of its outputs enter the interpretation of Burignat's results.

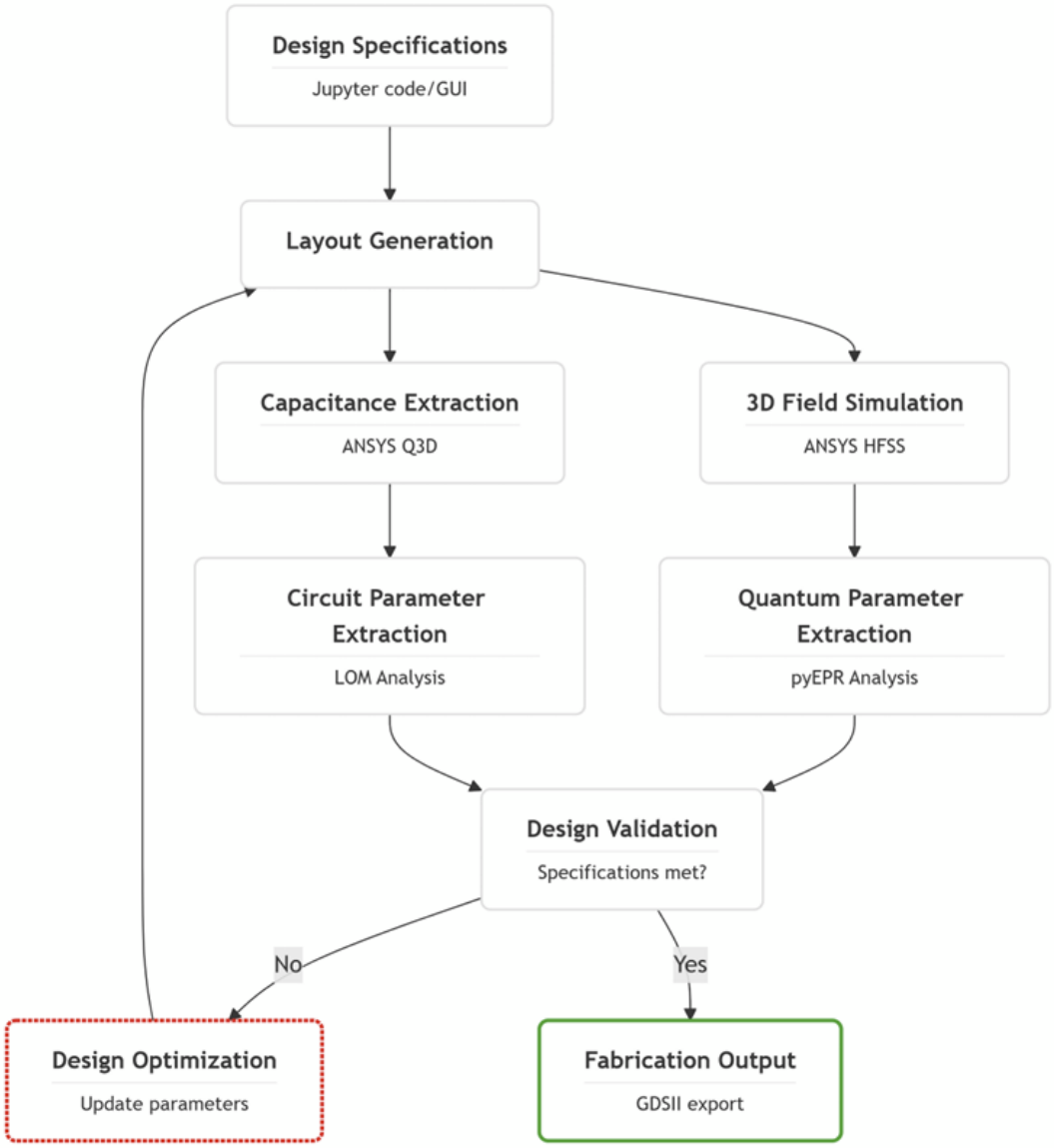


Figure 10: Integrated pre-fabrication optimization workflow followed in [22]. From a set of design specifications, a chip layout is generated with *qiskit-metal* and fed into two parallel simulation paths: Ansys Q3D capacitance extraction followed by lumped-oscillator-model (LOM) circuit-parameter extraction, and Ansys HFSS 3D field simulation followed by pyEPR quantum-parameter extraction. The extracted parameters are compared to the target specifications: if they match, the layout is exported as GDSII for fabrication; otherwise, the design parameters are updated and the loop is re-executed.

The workflow of Fig. 10, together with the Q3D-derived sensitivity maps and tolerance windows, constitutes a validated pre-fabrication methodology for a single, isolated transmon. It does not, however, address two ingredients that are central to a realistic processor: a charge line that capacitively drives the qubit, and inter-qubit couplers that enable two-qubit interactions between heterogeneous transmon variants. Filling these gaps is the starting point of the present work.

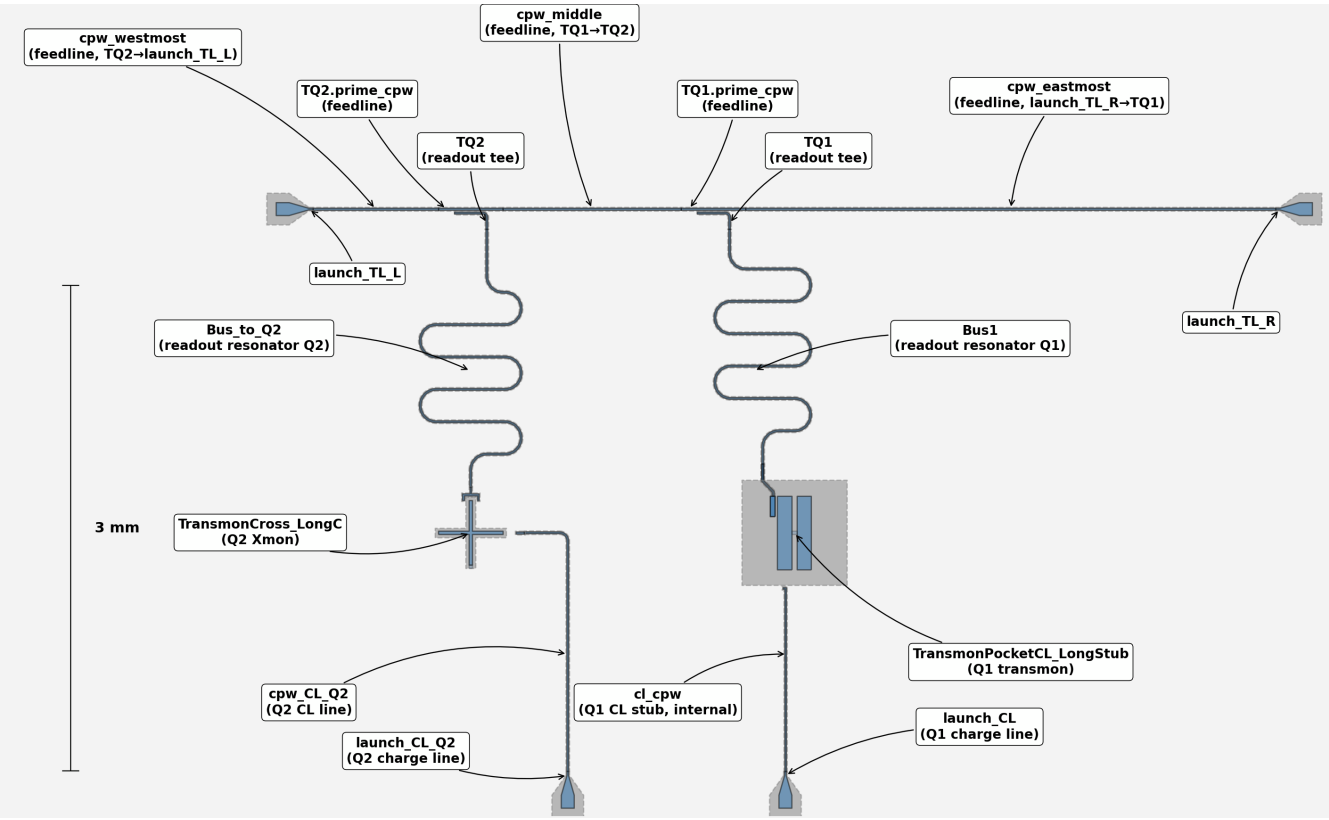


Figure 11: Reference two-qubit design studied in this work. A transmon-pocket qubit `Q1` (`TransmonPocketCL`, right) and an Xmon-class transmon `Q2` (`TransmonCross_LongC`, left) are each capacitively coupled to their own meandered $\lambda/4$ readout resonator (`Bus1`, `Bus_to_Q2`), terminated by the readout tees `TQ1` and `TQ2`. The resonator of `Q1` (`Bus1` plus its tee `TQ1`) is called *readout 1* throughout this work, and that of `Q2` (`Bus_to_Q2` plus `TQ2`) *readout 2*. The two tees are inserted on a common feedline: each `CoupledLineTee` internally realizes its own straight through-line (the "prime" segment), which is continued externally by the `RouteStraight` segments `cpw_eastmost`, `cpw_middle` and `cpw_westmost` — together forming the shared transmission line, launched at `launch_TL_L` and `launch_TL_R`, that acts both as the readout bus and as the sole mediator of the qubit–qubit interaction; no direct capacitive coupler links `Q1` and `Q2` in this configuration. Each qubit is equipped with a dedicated capacitive line for single-qubit drive: `Q1`'s line (`cl_cpw`) is routed entirely internally to the pocket down to `launch_CL`, while `Q2`'s line is routed externally (`cpw_CL_Q2`) to `launch_CL_Q2`.

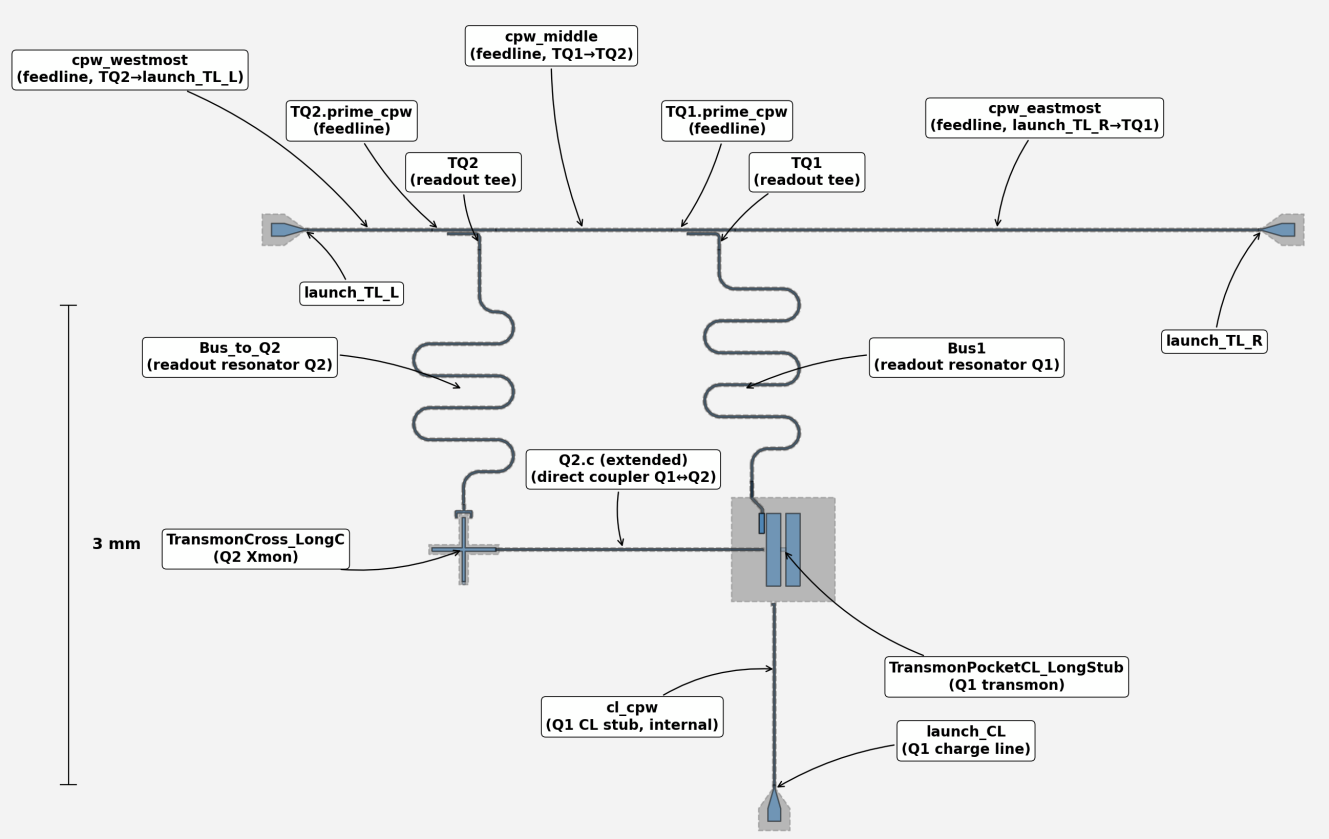


Figure 12: Second design studied in this work. The readout architecture (qubits, $\lambda/4$ resonators, tees and shared feedline) is identical to that of Fig. 11. The Xmon charge line `cpw_CL_Q2` has been removed and the corresponding cross arm of `Q2` is galvanically connected to the east arm, and extended horizontally (`Q2.c extended`) until it faces one capacitor pad of `Q1`, forming a fixed, always-on direct capacitive coupler between the two qubits. `Q1` retains its own charge line (`launch_CL` / `cl_cpw`); `Q2` lost its drive though the charge line.

Starting from Burignat's transmon-pocket chip, we add a second qubit — an Xmon-class transmon capacitively coupled to its own meandered $\lambda/4$ readout resonator — and equip each qubit with a dedicated charge line, required to deliver fast on-chip microwave XY pulses without routing them through the readout resonator [20]. Two variants of this two-qubit chip are then studied.

In the first design (Fig. 11), the transmon-pocket qubit `Q1` and the Xmon `Q2` interact only indirectly: their $\lambda/4$ resonators (`Bus1` and `Bus_to_Q2`) terminate on a shared feedline that simultaneously serves as the readout bus and as the only channel for qubit–qubit coupling. In the second design (Fig. 12), the Xmon charge line `cpw_CL_Q2` is removed and the corresponding cross arm is extended until it faces one of the transmon-pocket capacitor pads, forming a fixed, always-on direct capacitive coupler between `Q1` and `Q2`. This second layout provides a stronger, dedicated two-qubit interaction at the price of losing drive on `Q2`.

The central question addressed by comparing the two chips is whether inserting the direct coupler perturbs the local parameters of either qubit (transition frequency, anharmonicity, dispersive shift χ, qubit–resonator coupling g, and relaxation time T_1) with respect to the indirectly-coupled reference. In other words: can a direct coupler be dropped into an already validated single-qubit

geometry, or does its presence force a full re-optimisation of the pocket, of the cross, and of the two readout resonators? Answering this question is the object of the remainder of this thesis.

2 Geometry setup in Qiskit

2.1 Superconducting qubits

Before turning to the layout, we briefly recall what a superconducting qubit is. A superconducting qubit is a lithographic microwave circuit, cooled to ~ 20 mK, whose two lowest quantized energy levels are used as the computational states $|0\rangle$ and $|1\rangle$. We begin the representation with an LC oscillator circuit, a linear inductance L_r [21] and capacitance C_r Fig. 13(a). After quantization, it behaves as a quantum harmonic oscillator with equally-spaced levels of $\hbar\omega_r$ (Fig. 13(b)); a microwave drive on resonance excites every transition, leading to leakage out of the computational subspace. Anharmonicity is introduced by replacing the linear inductor with a Josephson junction (Fig. 13(c)), a non-dissipative non-linear inductance[22] realised by Cooper-pair tunnelling between two superconducting electrodes. The resulting cosine potential (Fig. 13(d)) has decreasing level spacing with energy, so that a microwave pulse can address the $|0\rangle \leftrightarrow |1\rangle$ transition alone. In the transmon regime [42], the junction is shunted by a large capacitance ($E_J/E_C \gg 1$), which exponentially suppresses sensitivity to charge noise at the cost of reduced anharmonicity, and yields the coherence times $\gtrsim 100$ µs reported in state-of-the-art devices. All qubits considered in this thesis are transmon variants.

[21] An inductance is linear when the induced voltage is proportional to the time derivative of the current, $V = L\,dI/dt$.

[22] An inductance is non-linear when L itself depends on the instantaneous current. For a Josephson junction the current–phase relation is $I = I_c \sin\varphi$, from which one derives the effective Josephson inductance $L_J(\varphi) = \Phi_0/(2\pi I_c \cos\varphi)$: it diverges as $\varphi \to \pi/2$ and grows with the oscillation amplitude. This current dependence turns the quadratic LC potential into a cosine, whose unequal level spacing gives the transmon its anharmonicity.

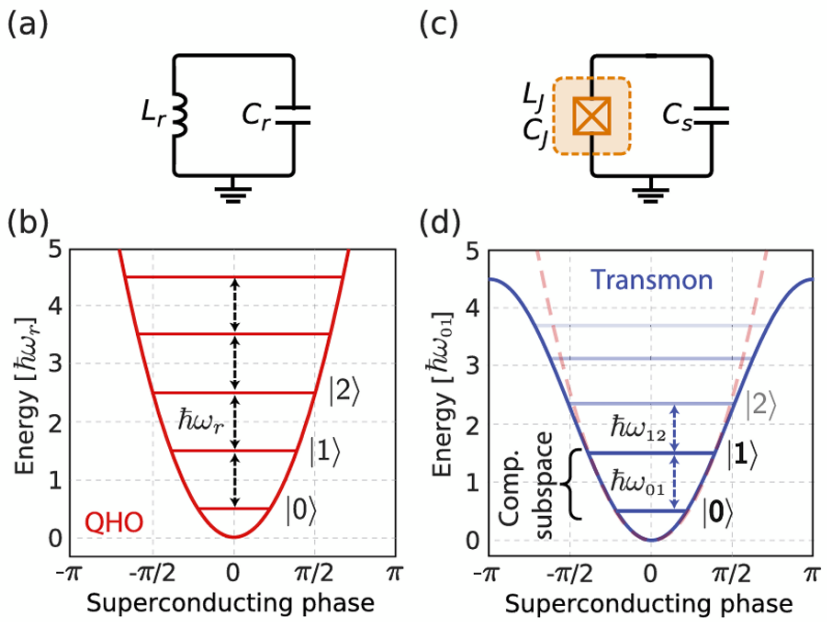


Figure 13: Comparison between the LC oscillator and the transmon, adapted from [22]. **(a)** LC oscillator with linear inductance L_r and capacitance C_r. **(b)** Quadratic potential of the resulting quantum harmonic oscillator, with equidistant levels separated by $\hbar\omega_r$: a resonant drive excites every transition. **(c)** Transmon circuit, in which the linear inductor is replaced by a Josephson junction (non-linear inductance L_J in parallel with junction capacitance C_J, dashed orange box) shunted by a large capacitance C_S. **(d)** Resulting cosine potential (solid blue) versus the harmonic potential (dashed red): the unequal level spacing isolates the $|0\rangle \leftrightarrow |1\rangle$ transition from higher ones.

This subsection introduces the qubit primitives and the auxiliary microwave components that make up the two designs of Section 1.3. All qubits considered in this work belong to the transmon family [42], but two complementary geometric realisations are used: the *transmon pocket*, in which two rectangular capacitor pads are recessed in a ground-plane pocket (Sec. 2.1.1), and the *Xmon*, in which the qubit island has a cross shape that exposes four orthogonal coupling ports (Sec. 2.1.2). Each qubit is equipped with a dedicated capacitive charge line for single-qubit XY control (Sec. 2.1.4), and the two-qubit interaction is mediated by the couplers described in Sec. 2.1.5. Every layout element is implemented through the parametric component library of *qiskit-metal*.

2.1.1 Transmon

The first qubit primitive is the transmon pocket, instantiated through the `TransmonPocketCL` class of *qiskit-metal*, the same transmon designed in Burignat's thesis where the same parameters values were kept but a charge line stub (Sec. 2.1.4) is added for a charge line connection (see Fig. 14). Its layout consists of two rectangular superconducting pads shunted by a single Josephson junction and recessed inside a rectangular cut-out of the ground plane (the "pocket"). The pocket is the defining feature of this geometry: by surrounding the qubit island with a close, well-defined ground reference, it concentrates the electric field inside a controlled volume, fixes the qubit's environmen-

tal impedance[23], and suppresses radiation into substrate modes[24] and stray on-chip capacitances.

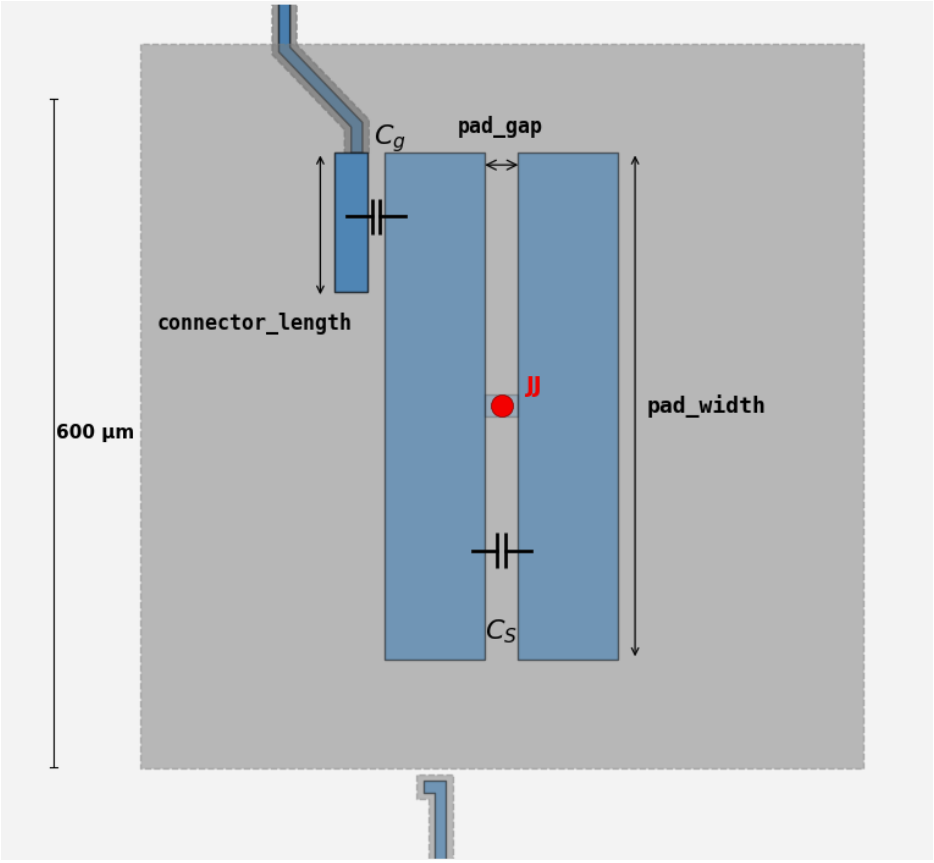


Figure 14: Layout of the transmon qubit showing two capacitor pads separated by `pad_gap` and connected via a JJ (red). Capacitive coupling to the resonator is set by `connector_length`, defining the geometric capacitance C_g. The shunt capacitance C_S arises from the pad geometry. The qubit resides in a metallic pocket defined by `pocket_width` and `pocket_height`, which controls parasitic coupling. Below is inserted a charge line stub for the charge line connection, see Sec. 2.1.4

The transmon Hamiltonian is set by only two energies: the Josephson energy E_J, which characterises the inductive non-linearity of the junction, and the charging energy

$$E_C = \frac{e^2}{2\,C_\Sigma}, \tag{20}$$

where $e = 1.602 \times 10^{-19}$ C is the elementary charge and $C_\Sigma = C_S + C_J + C_{\text{env}}$ is the total capacitance shunting the junction (C_S: shunt capacitance set by the qubit pads, C_J: intrinsic junction capacitance, C_{env}: stray capacitance to the surrounding environment, usually dominated by C_S). The role of the layout is to translate the desired ratio E_J/E_C — chosen in the transmon regime $E_J/E_C \gg 1$ — and qubit transition frequency

$$\omega_q \simeq \frac{\sqrt{8\,E_J E_C}}{\hbar}, \tag{21}$$

[23]The impedance $Z(\omega)$ seen by the qubit at its terminals when looking outward, which sets both the qubit frequency and the radiative decay rate via $\Gamma_1 \propto \mathrm{Re}[Y(\omega_q)]/C_\Sigma$.

[24]Electromagnetic modes of the dielectric substrate into which the qubit field can leak and lose energy irreversibly.

with $\hbar = h/2\pi$ the reduced Planck constant, into a concrete set of geometric parameters.

Four parameters dominate this translation and are exposed by `TransmonPocketCL`: `pad_gap`, which sets the inter-pad capacitance and, together with the pad area, the shunt capacitance C_S that dominates C_Σ; `connector_length`, the length over which the readout-resonator stub runs parallel to the pad, which sets the geometric coupling capacitance C_g between qubit and resonator and therefore the dispersive qubit–resonator coupling rate g; and `pocket_width`, `pocket_height`, which fix the distance to the surrounding ground plane and therefore the parasitic capacitances to ground (and, indirectly, the residual coupling to neighbouring elements). The Josephson energy itself is not a geometric parameter of the pocket: it is fixed by the junction's normal-state resistance through $E_J = \Phi_0 I_c/2\pi$, where $\Phi_0 = h/2e \approx 2.07 \times 10^{-15}$ Wb is the magnetic flux quantum and I_c is the junction critical current.

2.1.2 Xmon

The second qubit primitive used in this work is the Xmon, a cross-shaped transmon variant introduced by Barends *et al.* [54]. Electrically, the Xmon is still a transmon: a single Josephson junction shunted by a large capacitor in the regime $E_J/E_C \gg 1$, with the same $\omega_q \simeq \sqrt{8E_J E_C}/\hbar$ as the transmon pocket. The novelty is geometric: the capacitor island is shaped as a "+" sitting in a uniform ground-plane gap, so that each of the four arms exposes an independent capacitive coupling port. The qubit can therefore simultaneously host, on a single island, a readout-resonator coupling on one arm, a drive (charge) line on the second, and one or two inter-qubit couplers on the remaining arms, without crowding the layout. The flip side, compared with the transmon pocket, is a connectivity-for-shielding trade-off. The transmon-pocket geometry sits inside a closed cut-out of the ground plane that acts as a local shield: it fixes a clean impedance environment and suppresses parasitic capacitive coupling to neighbouring on-chip elements, at the price of allowing only one or two coupling ports to pierce the pocket. The Xmon removes that pocket and exposes the qubit island directly to a uniform ground gap (`cross_gap`), so up to four independent coupling ports become available on a single island; in exchange, parasitic coupling to neighbouring elements and to substrate modes is no longer shielded by a pocket and is controlled solely by `cross_gap`.

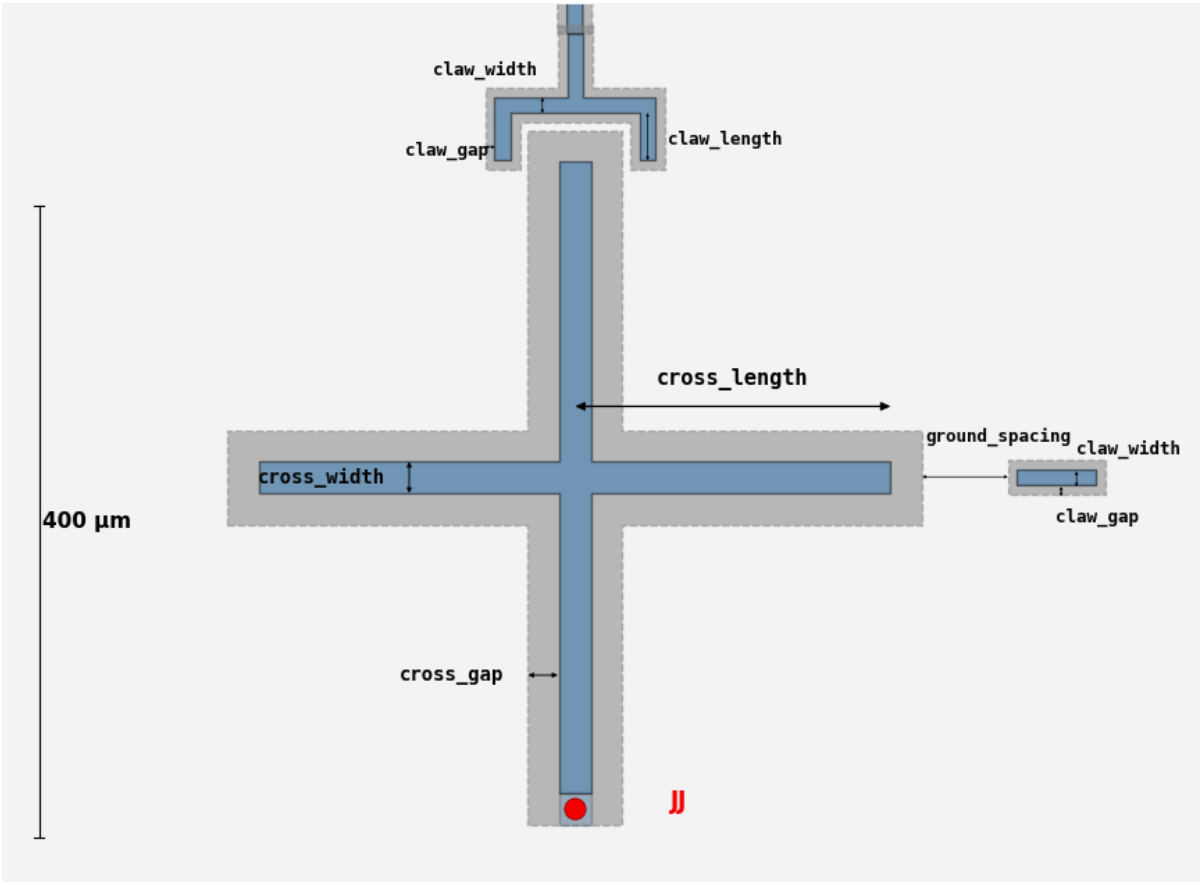


Figure 15: Parametric *qiskit-metal* layout of the Xmon used for `Q2`, with the geometric parameters exposed to the optimisation loop annotated. The Josephson junction (`JJ`, red dot) sits at the south-arm tip; the north arm (pad `b`) uses a claw connector (`connector_type = 0`), the east arm (pad `c`) a straight-gap connector (`connector_type = 1`). All parameters are defined in the surrounding text.

In *qiskit-metal* the Xmon is instantiated through the `TransmonCross` component (Fig. 15). Three geometric parameters set the cross itself and, through it, the qubit Hamiltonian. In contrast to the transmon pocket, where the shunt capacitance C_S arises from the mutual capacitance between two rectangular pads, the Xmon has no pad pair: the "+"-shaped island floats in a uniform ground-plane gap, and C_S is the capacitance between the cross and the surrounding ground plane [54]. It is controlled by `cross_length`, the half-length of each arm from the centre of the cross to its tip, which sets the total metal area facing ground and provides the dominant contribution to C_S and therefore to E_C and ω_q; by `cross_width`, the lateral width of each arm, which fine-tunes C_S and the characteristic impedance seen at each coupling port; and by `cross_gap`, the uniform dielectric gap between the cross metal and the surrounding ground plane. As for the transmon pocket, the junction is not a geometric parameter of the cross and enters the simulation as the Josephson inductance $L_J = \hbar/(2eI_c)$.

Each of the four arms can host up to one *connector pad* that carries the capacitive coupling to an external element (readout resonator, charge line, or another qubit). Its azimuthal position on the cross is fixed by `connector_location` $\in \{0°, 90°, 180°, 270°\}$, corresponding to the east, north, west and south arms. Two pad topologies are exposed by the `connector_type` option, and both

appear in Fig. 15: `connector_type = 0` produces a C-shaped *claw* that wraps around the arm tip (north arm, pad `b`) and is used when the incoming coplanar waveguide (CPW) terminates without galvanic contact - typically the readout coupling; `connector_type = 1` produces a straight *gap* pad (east arm, pad `c`), used when the incoming CPW makes direct metallic contact with the pad, as is the case for the charge line `cpw_CL_Q2` of `Q2`. In both cases the pad dimensions are controlled by the same four parameters: `claw_length`, its extent along the arm; `claw_width`, its metal width; `claw_gap`, the dielectric gap that isolates it from the surrounding ground plane; and `ground_spacing`, the distance between the cross-arm tip and the pad. Together they set the coupling capacitance between the qubit island and the external CPW, and are the primary knobs used to tune the qubit–resonator coupling g or the effective drive amplitude of the charge line.

2.1.3 Readout resonator

The readout is not redesigned in this work: the $\lambda/4$ resonator of Ref. [22] is taken as it stands and instantiated twice, once per qubit. It is built from two stock components. A `CoupledLineTee` carries the shared feedline on its `prime` line and, on its `second` line, the end of the resonator that runs parallel to it over `coupling_length` $= 200$ µm at a gap D (Fig. 16); the option `open_termination = False` leaves that end galvanically shorted to the ground plane, which places a voltage node there and makes the line a quarter-wave. A `RouteMeander` of `total_length` $= 4.1$ mm (`fillet` $= 99$ µm) then carries the line from the tee to the qubit, where it stops on a connector pad and is coupled capacitively to the island. The two pieces together form one 4.258 mm $\lambda/4$ line, designed in Ref. [22] for $\omega_r/2\pi = 7$ GHz, i.e. a detuning of 1.5–2.5 GHz from a qubit at 4.5–5.5 GHz.

Both qubits carry a copy of that pair, and the component names used throughout this work follow *Qiskit Metal*: `TQ1` + `Bus1` reads Q_1, landing on the pocket connector pad `a`; `TQ2` + `Bus_to_Q2` reads Q_2, landing on the north claw of the Xmon. Only the placement of the tee on the feedline differs, at $(-0.5, 2)$ mm and $(-2, 2)$ mm. The geometry of the coupling zone, the resonator length and the feedline cross-section are identical in both branches, so the two readouts are nominally degenerate.

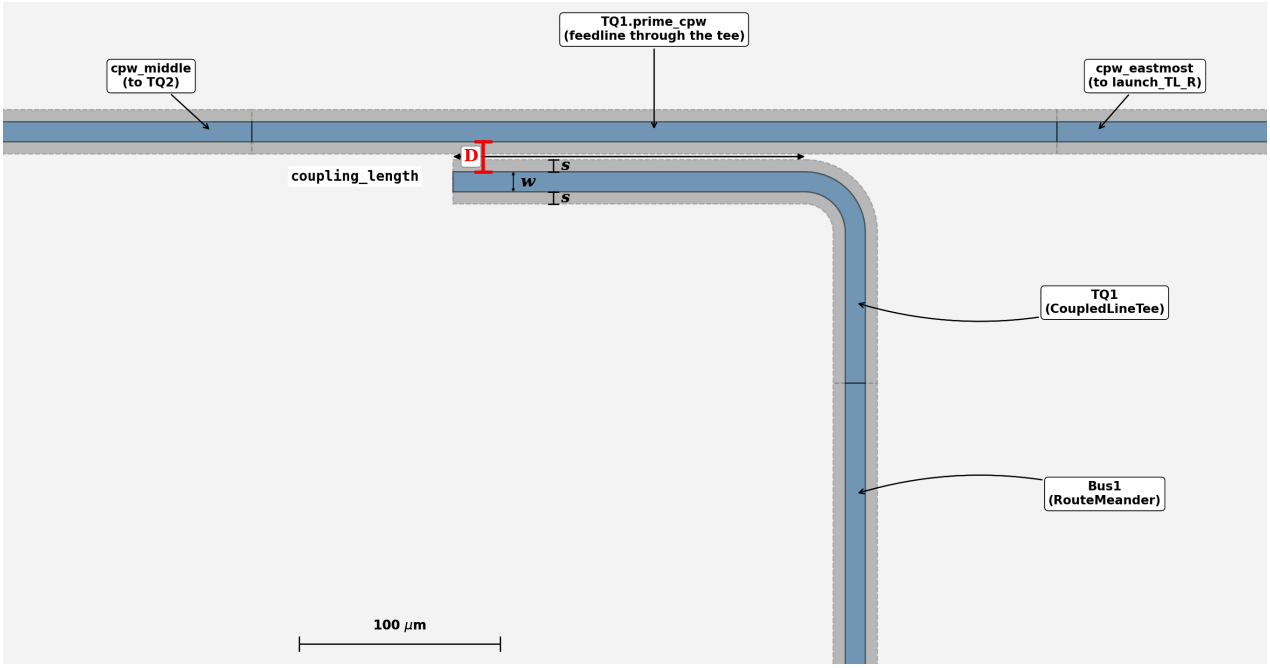


Figure 16: Feedline–resonator coupling zone of Q_1, reproducing Fig. 19 of Ref. [22]: centre-conductor width w, ground separation s, `coupling_length` and coupling gap D. The boxes name the components: the feedline is the `prime` line of `TQ1`, whose `second` line becomes the meander `Bus1` at the pin `second_end`. `TQ2` + `Bus_to_Q2` are identical.

2.1.4 Charge line

TransmonPocket (`Q1`) To perform single-qubit XY gates, each qubit needs a dedicated microwave drive port that couples capacitively to the qubit island without loading it enough to spoil T_1 [20]. For the transmon-pocket qubit `Q1`, this port is provided by the integrated *charge-line stub* of the `TransmonPocketCL` component of *qiskit-metal*: an L-shaped CPW carved into the pocket ground plane, whose open end (pin `Charge_Line`) is subsequently wired to an external launch pad and, through it, to the room-temperature drive electronics.

The stub geometry, shown in Fig. 17, is fully parametric. `cl_width` and `cl_gap` set the CPW cross-section (centre-conductor width and dielectric gap to ground) and therefore its characteristic impedance Z_0; both are fixed here at 10 µm and 6 µm to match a $\simeq 50\ \Omega$ line[25]. `cl_length` controls the short arm of the L (the segment parallel to the pocket edge, 20 µm in `Q1`), which sets the geometric coupling capacitance between the stub tip and the nearest transmon pad. `cl_ground_gap` is the dielectric gap between the stub and the pocket edge (6 µm), preventing the stub metal from shorting the pocket boundary while keeping the pocket a good local shield. Finally, `cl_off_center` breaks the horizontal symmetry of the pocket by shifting the stub laterally (50 µm in `Q1`) so that it faces the pad it is meant to drive.

[25] The entire drive chain (cables, attenuators, wirebonds, on-chip CPWs) is designed for a reference impedance of 50 Ω. Any impedance mismatch reflects part of the incident microwave power before it reaches the qubit and distorts the pulse shape. Matching the on-chip CPW to 50 Ω ensures that the intended drive is faithfully delivered to the qubit.

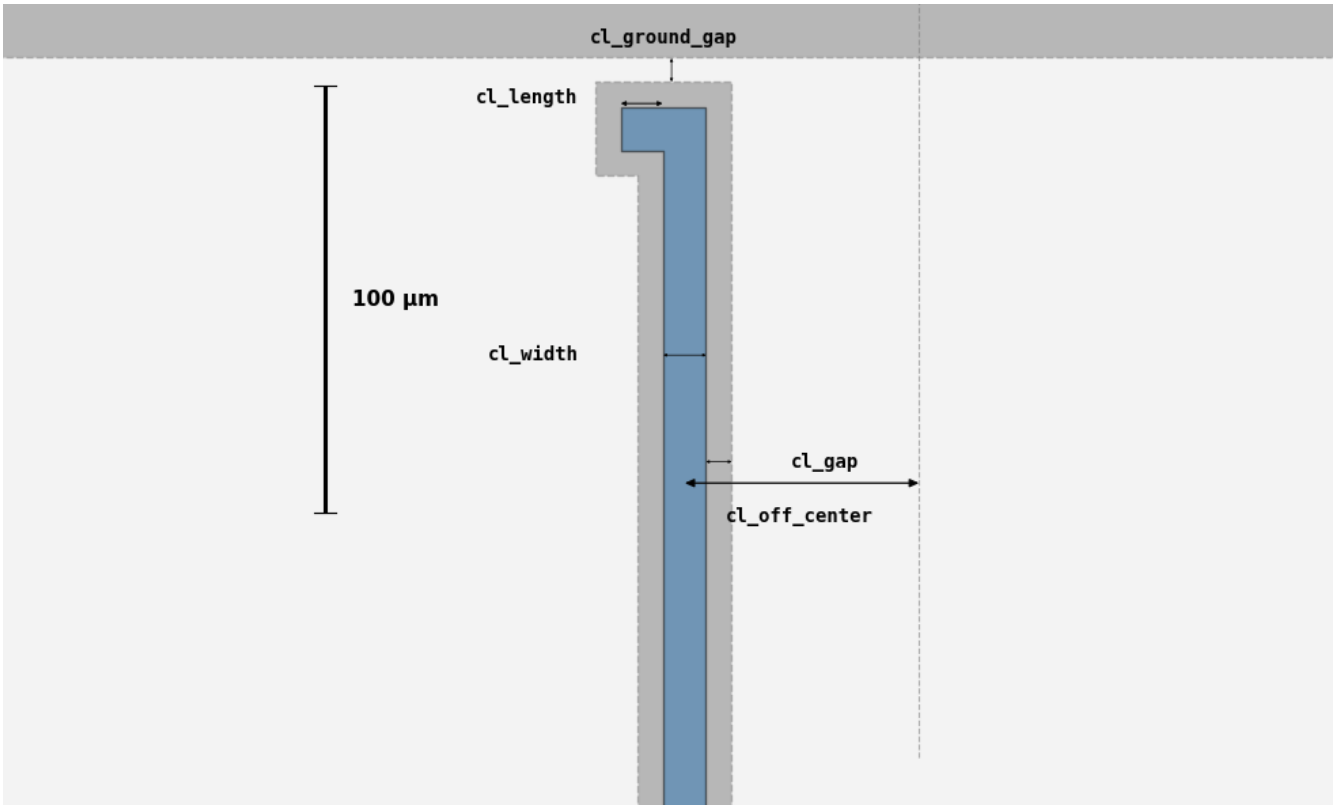


Figure 17: Zoom on the charge line stub, the internal sub-geometry of `TransmonPocketCL` that capacitively couples the external drive line to the qubit island. The L-shaped CPW consists of a short arm of length `cl_length` (parallel to the pocket edge) and a longer trace of width `cl_width` that terminates at the `Charge_Line` pin. The dielectric gap `cl_gap` between the center conductor and the surrounding ground plane sets the characteristic impedance, while `cl_ground_gap` is the dielectric gap that isolates the stub from the pocket ground plane where it exits the pocket. `cl_off_center` breaks horizontal symmetry by introducing a horizontal offset of the cl stub.

Extended-stub subclass. The parent `TransmonPocketCL` class hard-codes the horizontal segment of the L to $8 \times$ `cl_width` ≈ 80 µm. In our layout that would make the `Charge_Line` pin land only a few tens of µm below the pocket etcher, so any external CPW connecting the pin to the launchpad would have to run underneath the pocket etch region, producing an overlap between the pocket cut-out and the drive-line cut-out. To eliminate it, we defined a subclass `TransmonPocketCL_LongStub` that exposes the horizontal-segment length as a new option `cl_cpw_length`, independent of `cl_width`. Extending the stub in this way pushes the pin far below the pocket boundary without changing Z_0, since `cl_width` and `cl_gap` are untouched. Setting `cl_cpw_length` $= 1.138$ mm on `Q1` places the `Charge_Line` pin directly on the tie of the wirebond launchpad `launch_CL` at $(-0.055, -1.437)$ mm, so that the internal stub reaches the launchpad on its own and no external `cpw_CL` route is required. `Q1` is therefore driven through a single continuous CPW that runs from the launchpad to the pocket edge without ever crossing another etched region. The result is shown in Fig. 19

Xmon (`Q2`). The Xmon has no built-in charge-line stub: the only capacitive ports available are the four arms of the cross itself. The drive is therefore delivered through one of the connector pads introduced in Sec. 2.1.2, chosen with `connector_type = 1` (straight-gap topology) on the

east arm (`connector_location = 180`) (see Fig. 18). Topologically this pad plays the same role as the readout claw on the north arm: it is a small metal island, isolated from the Xmon cross by a dielectric gap, and galvanically continuous with the external CPW attached to it through the pin `Q2.c`. Pad `c` therefore forms a single continuous piece of metal with `cpw_CL_Q2` all the way to the launchpad `launch_CL_Q2`, and the qubit is driven capacitively across the ground gap that separates the pad from the cross, exactly as it is read out capacitively across the corresponding gap on the north arm. The only difference between the two pads is their shape: a wraparound C on the north arm, a straight bar on the east arm.

The pad geometry uses the same four parameters as any other connector pad: `claw_length`, `claw_width`, `claw_gap` and `ground_spacing`. For `Q2` these are set to 30 µm, 10 µm, 6 µm and 54 µm, so that the total dielectric gap between the pad and the cross along the drive direction is `cross_gap` + `ground_spacing` + `claw_gap` = 20 + 54 + 6 = 80 µm — the same gap-to-ground as on the readout side of the cross, so that the local impedance environment stays uniform around the qubit.

Extended-arm subclass. The parent `TransmonCross` class hard-codes the length of the metallic arm of a `connector_type = 1` pad to 4 × `claw_width` = 40 µm. Since the pin `Q2.c` sits at the east tip of this arm, and since `cpw_CL_Q2` starts exactly at `Q2.c`, this fixes both the extent of the pad and the starting point of the drive CPW. In practice we want independent control over where the drive line leaves the qubit, in order to accommodate a launchpad placed at a fixed position on the chip carrier, or to obtain a specific CPW routing that avoids other components. We therefore defined a subclass `TransmonCross_LongC` that exposes the arm length as a new per-pad option `gap_extend_length`: increasing it shifts the pin, and with it the west end of `cpw_CL_Q2`, east by the same amount, keeping pad `c` and the CPW as a single continuous conductor. For `Q2` we set `gap_extend_length` = 50 µm (Fig. 18); all other pads including the readout pad `b` keep the parent behaviour. The final layout is highlighted at Figure 19.

Finally, an etcher overlap is visible on the east arm in Fig. 18, where pad `c` meets the incoming drive CPW `cpw_CL_Q2`: the pad and the CPW are drawn by two different components, each with its own etcher, and these etchers overlap where the metals meet at the pin. The same artefact appears on the north arm at the junction between pad `b` and the readout meander `Bus_to_Q2`. Both overlaps are harmless for Ansys Q3D and Ansys HFSS as both solvers correctly interpret the union of the overlapping etch regions, so we leave them in place. They show up only as an artefact in the layout in *Qiskit* and never enter the extracted Hamiltonian parameters.

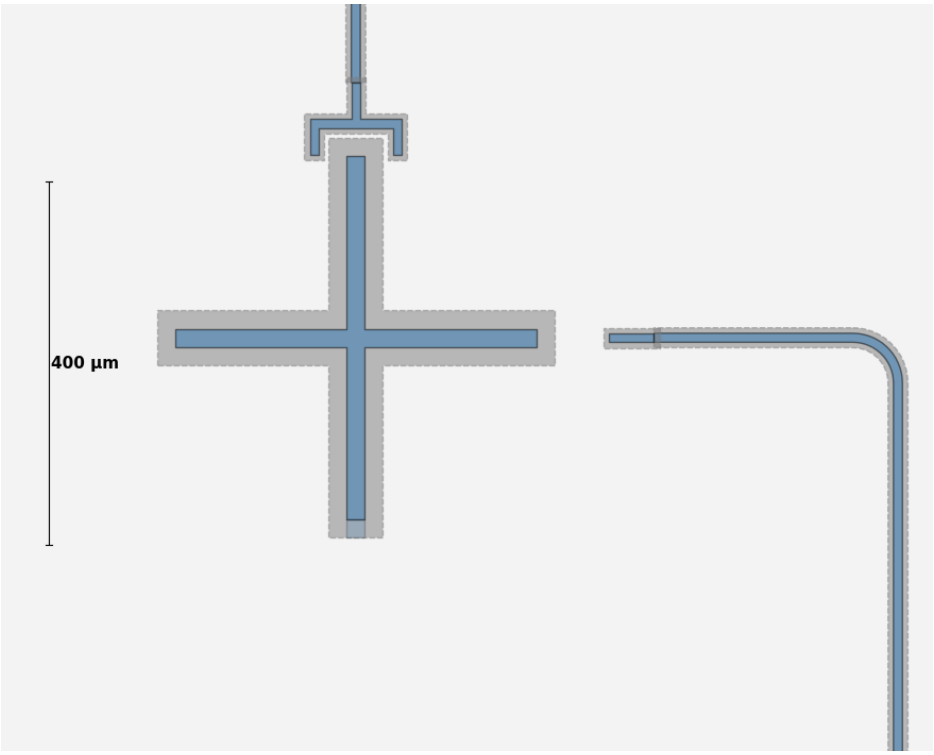


Figure 18: Zoom on pad `c` of the Xmon `Q2` (east arm) and the incoming drive CPW `cpw_CL_Q2`. The pad uses `connector_type = 1` (straight-gap topology): it is a small metal island, isolated from the Xmon cross by a ground gap (`cross_gap` + `ground_spacing` + `claw_gap` = 80 µm), and galvanically continuous with `cpw_CL_Q2` through the pin `Q2.c`. The arm length is set by `gap_extend_length` = 50 µm through the subclass `TransmonCross_LongC`, which fixes the position of `Q2.c` and therefore the west end of `cpw_CL_Q2`. The qubit is driven capacitively across the pad-to-cross ground gap.

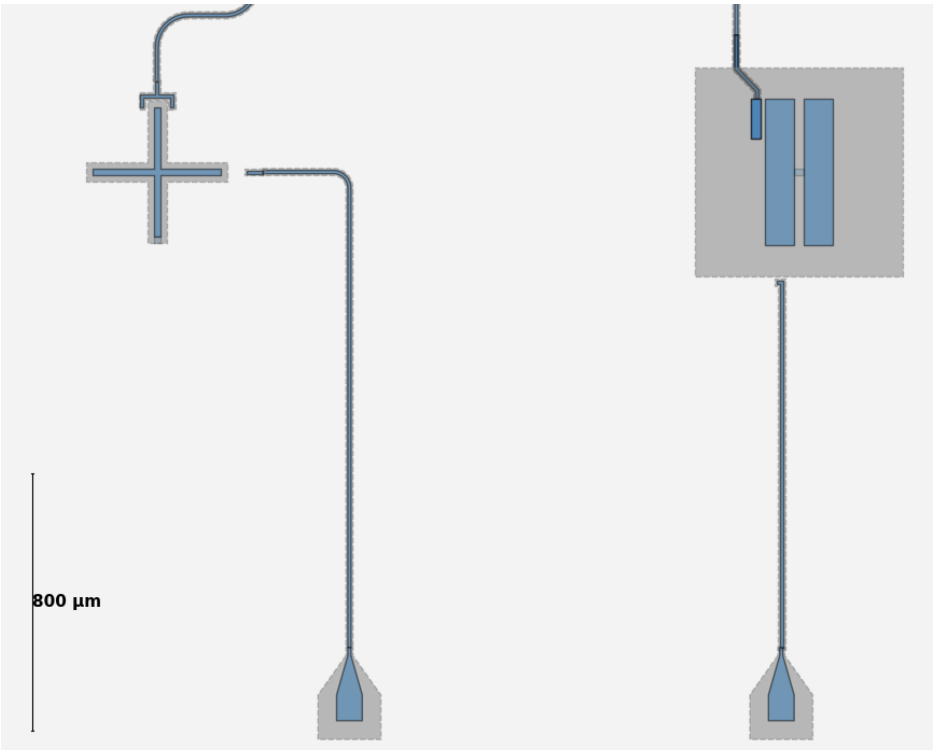


Figure 19: Overview of the two charge lines in the indirect-coupling model. On the right, the transmon-pocket qubit `Q1` is driven through its integrated stub, extended by `cl_cpw_length = 1.138 mm` via `TransmonPocketCL_LongStub` so that the `Charge_Line` pin lands directly on the wirebond launchpad `launch_CL` without any external coplanar waveguide (CPW). On the left, the Xmon `Q2` is driven through pad `c` on its east arm (`connector_type = 1`), which is galvanically bonded to the external CPW `cpw_CL_Q2`; this CPW then routes south to the launchpad `launch_CL_Q2`. The two drive paths therefore illustrate the two distinct integration strategies described in Sec. 2.1.4: an internally-routed stub for the pocket and an externally-bonded pad for the Xmon.

2.1.5 Couplers

The direct-coupler approach requires bringing the two qubits into close proximity, for reasonable values of C_{12}. In *Qiskit Metal*, one way to achieve it is by extending the connector pad `c` of the Xmon (`TransmonCross`) toward the `TransmonPocketCL`. In the indirect-coupling design, this same pad served as Q2's charge/drive line; here it is repurposed and extended, using the same technique as for Q1's charge-line stub (overriding the parent component's hard-coded arm length via subclassing), to instead form the direct capacitive coupler. The gap between the extended arm and the `TransmonPocketCL` pad is set by `gap_extend_length` = 1.674 mm, together with `ground_spacing` = −26 µm which force arm-connector galvanic contact, thus jointly controlling C_{12}. Fig. 20 shows the resulting layout.

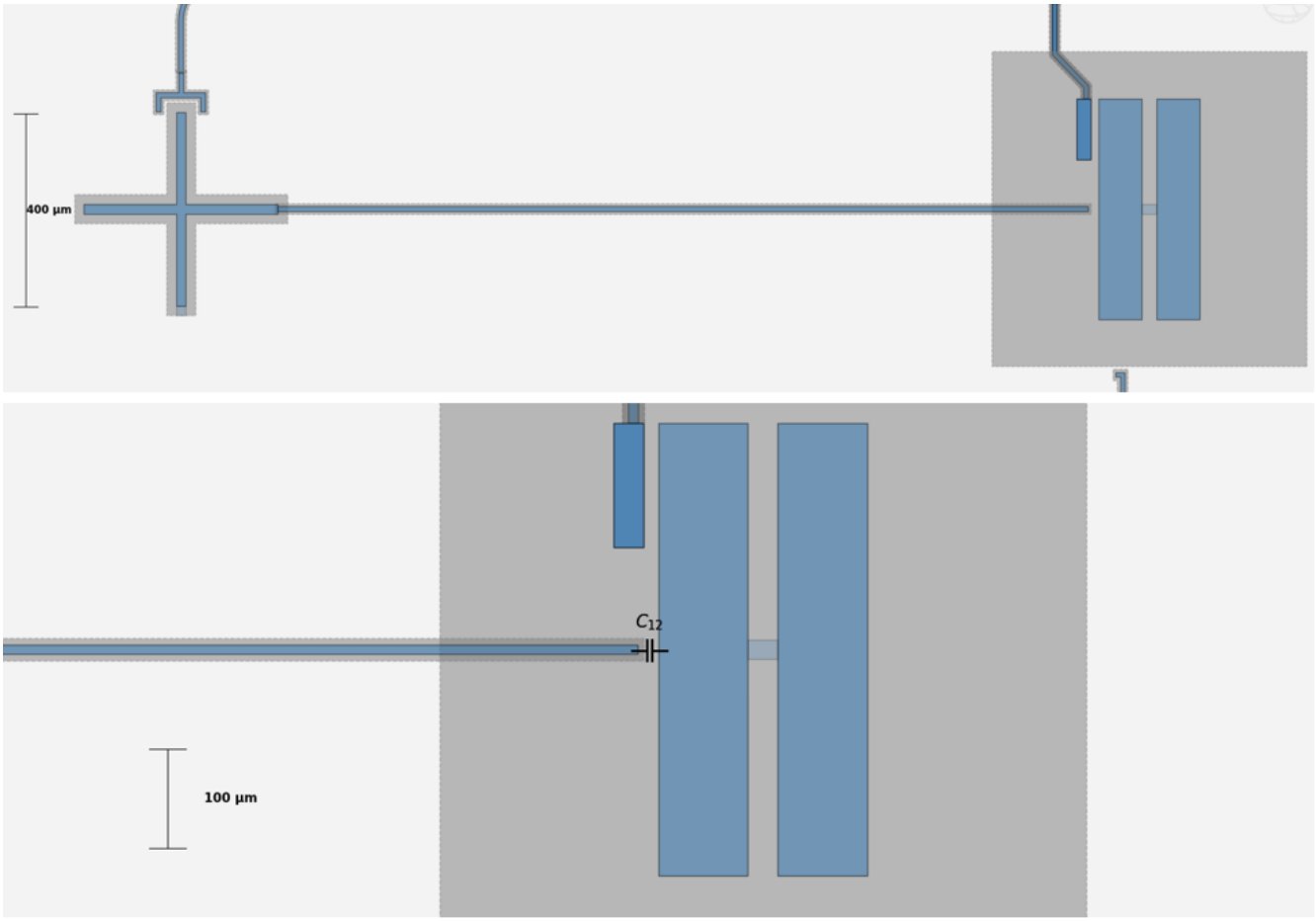


Figure 20: Direct coupling gap between `Q1` (`TransmonPocketCL`, right, gray pocket) and `Q2` (`TransmonCross_LongC`, left, Xmon). **(Top)** Full extent of the coupler arm, from the Xmon cross to `Q1`'s pocket, with the cross conductor width visible on `Q2`'s pad `c`. The pad is extended toward `Q1` by `gap_extend_length`, together with a negative `ground_spacing` $= -26\,\mu$m that forces galvanic contact between the arm and its back connector. **(Bottom)** Close-up of the resulting gap, stopping short of `Q1`'s west pad edge to leave a non-galvanic capacitive gap; the coupling capacitance C_{12} is indicated at the gap location. No external route or polygonal coupler is instantiated — the extended arm itself is the coupler.

2.1.6 Resulting lumped-element circuit

Both layouts can finally be reduced to a circuit. At 6 GHz, a signal propagating on the chip has a wavelength of about 2 cm. The qubit pads, the coupling gaps and the charge-line stubs are at most a few hundred micrometres long, hence far shorter than this wavelength: the voltage is then the same everywhere on such a piece of metal, which can be treated as a single node, and two neighbouring nodes simply as a capacitor. The readout resonators and the feedline, by contrast, are millimetres to centimetres long, i.e. comparable to the wavelength, so the voltage does vary along them. Each resonator is however used only around its own $\lambda/4$ resonance, where it is equivalent to a parallel $L_r \| C_r$ circuit and is drawn as such; the feedline is kept as a transmission line, closed at both ends by its matched 50 Ω loads. What is left is a network of capacitances closed by the two Josephson junctions (Figs. 21 and 22). This is the circuit that Q3D fills with numbers (Sec. 3.2) and that the LOM quantises (Sec. 3.2.3).

Comparing the two figures shows what the coupler changes. The readout side is the same, so the only difference is how the two qubits see each other. In Model 2, C_{12} connects the two islands directly, which makes the coupling first order and fixed by the coupler geometry alone. At the same time, Q_2 loses its charge line, hence its C_{d2} branch, so the capacitance seen by Q_2 – and therefore its E_C and its frequency – is not necessarily the same in both models. Measuring this shift is the object of the next section.

Model 1 deserves a separate comment, because it is not the indirect architecture described in Sec. 1.1.8. In the canonical bus scheme of Ref. [44], both qubits are capacitively connected to the *same* resonator, and eliminating that single mode by a Schrieffer–Wolff transformation to second order in g_j/Δ_j produces the effective coupling of Eq. (12). Here the two qubits share no such mode: Fig. 21 shows that the only path linking them is the chain C_{g1} – readout 1 – C_{k1} – feedline – C_{k2} – readout 2 – C_{g2}, i.e. three mediating elements in series rather than one. An excitation must therefore be virtually created and destroyed at each of them, and the exchange survives only at fourth order in perturbation theory,

$$g_{12}^{\mathrm{eff}} \sim \frac{g_1\, g_{k_1}\, g_{k_2}\, g_2}{\Delta_{r_1}\, \Delta_f\, \Delta_{r_2}}, \tag{22}$$

where g_1 and g_2 are the qubit–readout couplings, g_{k_i} the readout–feedline couplings, and Δ_{r_i}, Δ_f the detunings of the intermediate modes from the qubit being excited. Each additional link multiplies the result by a further factor g/Δ, typically 10^{-2} for the geometries considered here, so Eq. (22) is smaller than Eq. (12) by roughly four orders of magnitude for the same qubit–resonator couplings.

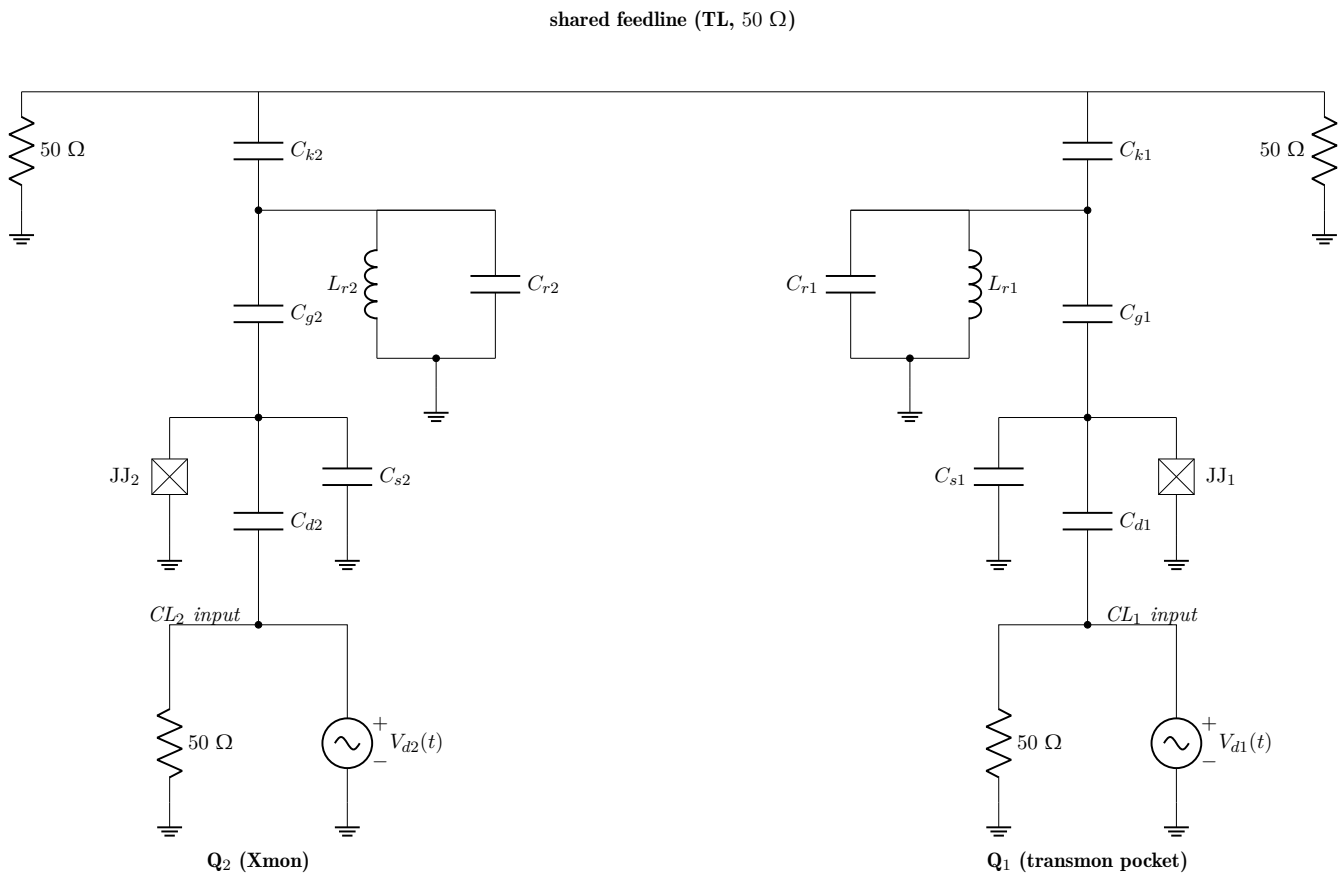


Figure 21: Lumped-element equivalent of the indirect-coupling model. Each column reads top to bottom: coupling capacitance $C_{k,i}$ to the shared feedline, $\lambda/4$ readout resonator $L_{r,i}\|C_{r,i}$, resonator–qubit coupling $C_{g,i}$, qubit island (shunt capacitance $C_{s,i}$ in parallel with the Josephson junction JJ_i), drive coupling capacitance $C_{d,i}$, and the charge line modelled by its matched 50 Ω impedance and its microwave source $V_{d,i}(t)$.

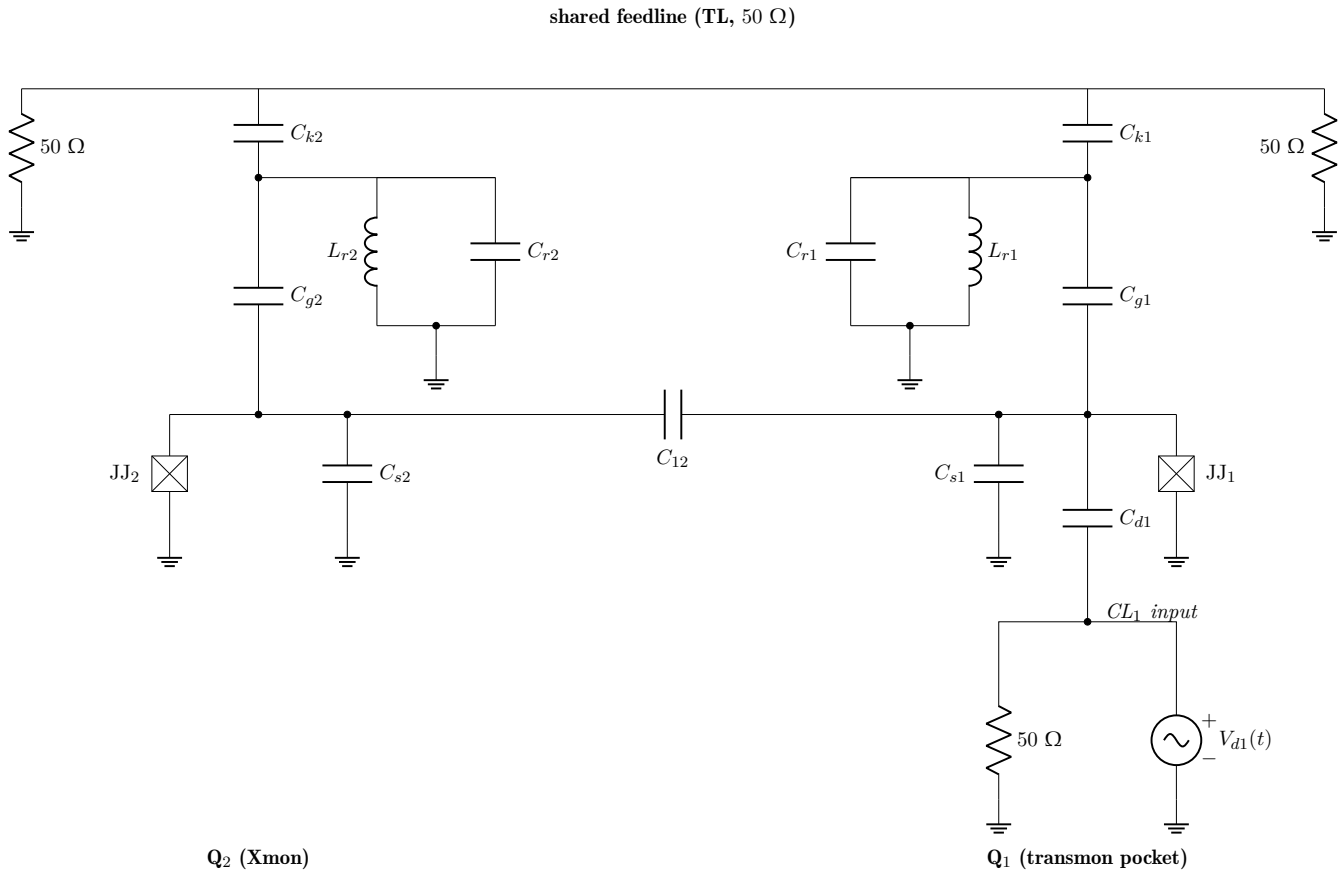


Figure 22: Lumped-element equivalent of the direct-coupling model. The readout side is identical to Model 1 (Fig. 21): each qubit is capacitively coupled through $C_{g,i}$ to its own $\lambda/4$ readout resonator $L_{r,i}\|C_{r,i}$, itself coupled through $C_{k,i}$ to the shared 50 Ω-matched feedline. The charge line of Q_2 has been removed and replaced by a fixed direct capacitive coupling C_{12} between the two qubit islands; only Q_1 retains a drive line (matched 50 Ω impedance and source $V_{d1}(t)$).

3 Simulation

3.1 High-Frequency Structure Simulator (HFSS) eigenmode solver and energy-participation-ratio (EPR) post-processing

This section runs the Ansys HFSS eigenmode solver on the two designs of Sec. 2 in order to extract the five eigenfrequencies of the system, together with their junction participation ratios p_{jj,Q_i}[26] and their dielectric participation ratio p_{diel}[27]. The workflow reuses the pipeline developed in Burignat's thesis [22], but with an important difference: Burignat exploited the HFSS solver purely as a modal validation step (checking that the intended qubit, resonator and feedline modes were correctly localised on the expected metal features). We go one step further: the extracted eigenfrequencies and participation ratios are compared between the two designs (Sec. 3.1.3) and used to discuss the parameter changes with respect to Burignat's single-transmon reference chip.

We recall the working principle of the simulation following Burignat's description [22]. HFSS is a full-wave finite-element solver that discretises the 3D chip geometry (metal, substrate and vacuum box) into a tetrahedral mesh and solves Maxwell's equations in the frequency domain. In eigenmode mode, no external port excitation is applied: the solver instead searches for the N lowest resonant frequencies at which the passive structure supports self-sustained standing electromagnetic modes. Each Josephson junction is linearised beforehand as a lumped $L_j \parallel C_j$ element (with $L_j = 10$ nH and $C_j = 2$ fF throughout this work), so that the junction contributes a linear inductance to the modal Hamiltonian while its non-linearity is deferred to the pyEPR post-processing. The mesh is refined iteratively through an adaptive-pass loop: at each pass, the mesh is locally densified where the modal fields display the largest gradients, until the relative shift of every eigenfrequency between two successive passes falls below 0.5%. Once converged, HFSS returns for each mode its eigenfrequency and its full 3D electric field distribution; pyEPR then post-processes these fields to extract the junction p_{jj,Q_i} and substrate energy participation ratios $p_{\text{diel,main}}$. The full pyEPR methodology and analytical derivation of the HFSS simulation is explained in Ref. [22]

Through this work, both models are solved with the same HFSS setup, summarised in Table 3: $N = 5$ eigenmodes requested (one per resonant element of the chip: the two qubits, their two $\lambda/4$ readout resonators and the shared feedline (Sec. 3.1.3)) with a convergence target of $\max_m |\Delta f_m / f_m| < 0.5\%$ maintained over three consecutive passes, a minimum of eight passes and a hard ceiling of 18 adaptive passes.

[26] The junction participation ratio p_{jj,Q_i} is the fraction of a mode's total inductive energy stored in the Josephson junction of qubit Q_i.

[27] The dielectric participation ratio $p_{\text{diel},k}$ is the fraction of a mode's total electric energy stored in dielectric region k; here we retain only $p_{\text{diel,main}}$, the participation in the silicon substrate.

Parameter	Value
Number of modes	5
Convergence criterion	$\max_m \lvert\Delta f_m/f_m\rvert < 0.5\%$
Minimum adaptive passes	8
Maximum adaptive passes	18

Table 3: HFSS eigenmode solver settings used for both Model 1 and Model 2, following Burignat's specifications [22]. Only N was updated.

3.1.1 Convergence of the adaptive solves

Figure 23, **(top)** shows the eigenfrequencies convergence relative to the pass number of the indirect coupling model. The convergence criterion of $\max_m \lvert\Delta f_m/f_m\rvert \leq 0.5\%$ was met at the 18th pass (Fig. 23, **(bottom)**), indicating a successful simulation with low uncertainty, i.e. low margin of error. However, modes 3 and 4 remain spectrally close throughout the refinement, which is the signature of a hybridised qubit-resonator pair; this near-degeneracy is discussed in 3.1.4.

Figure 24 shows the same analysis for the direct-coupling model, and the outcome is qualitatively different: the criterion is never met and the solve stops on the hard ceiling of 18 passes (Fig. 24 (**bottom**)). The maximum relative shift falls quickly during the first passes but then plateaus between 2% and 8%, an order of magnitude above the 0.5% target, and still stands at about 2% at the last pass. The mode responsible is mode 1, the Xmon: from pass 2 onwards its eigenfrequency increases monotonically, from 1.1 to 2.30 GHz, and it is still rising when the solver stops, so the returned 2.295 GHz has to be read as a lower bound rather than as a converged value. Modes 2–5, on the contrary, flatten after roughly the tenth pass exactly as in Model 1, so the difficulty is localised on the Xmon and does not contaminate the pocket, the two readout resonators or the feedline. This is consistent with the only geometrical difference between the two models: the extended cross arm adds a large metal area facing the ground plane and the pocket, which increases the total capacitance of Q_2 and pushes its mode down in frequency. Whether the remaining drift is numerical or physical, and what it implies for the extracted parameters, is examined in Sec. 3.2.3 and Sec. 3.3.

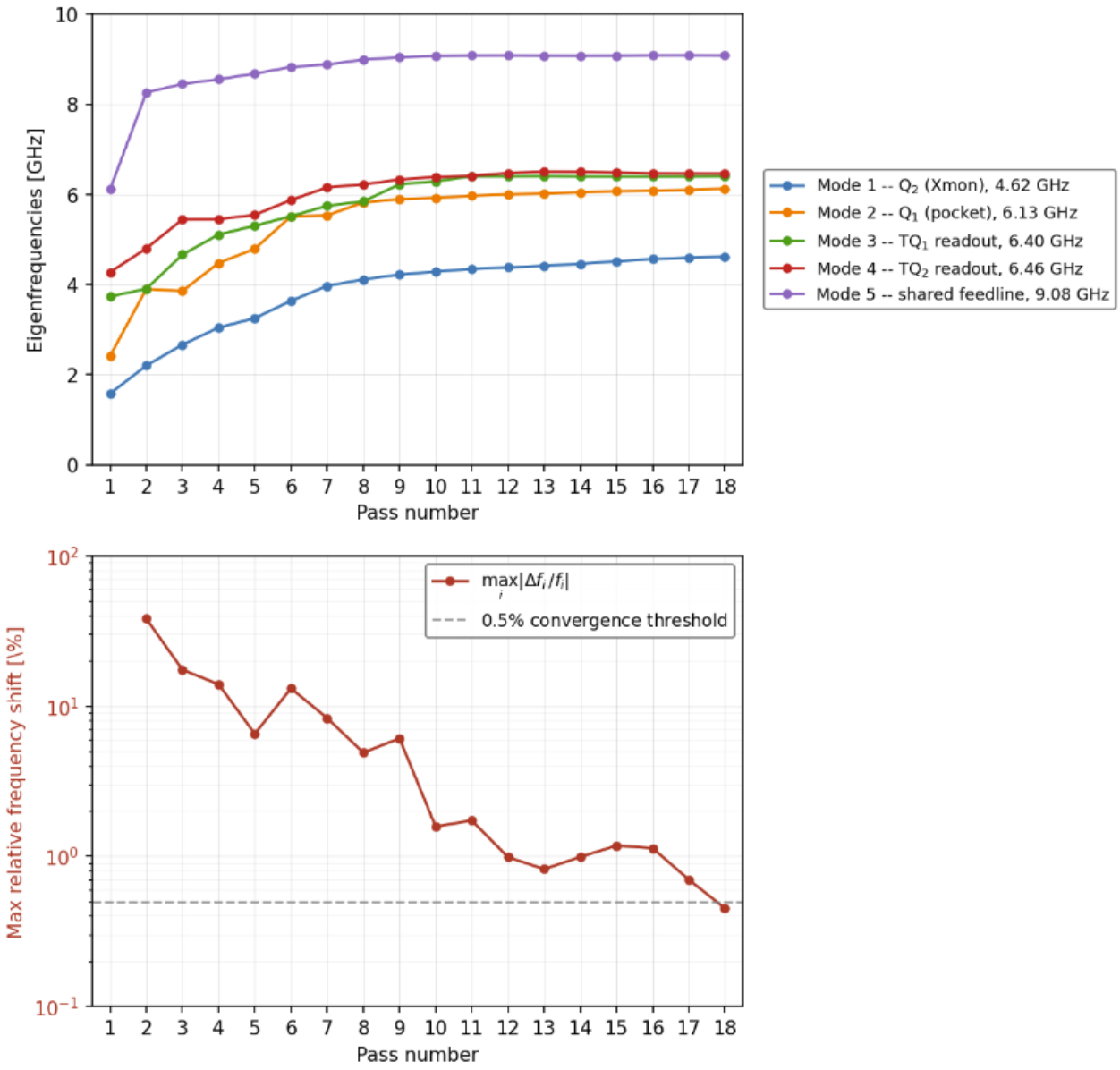


Figure 23: Convergence of the Model 1 full-chip HFSS eigenmode analysis over 18 adaptive passes. **(top)** Frequency of each of the five eigenmodes as a function of the pass number. **(bottom)** Maximum relative frequency shift ($\max_m |\Delta f_m / f_m|$) between consecutive passes. The 0.5% threshold target is crossed at the final passes, indicating that the high frequency mesh solver has resolved the five modes with a low uncertainty. In the legend, "TQ_1 readout" and "TQ_2 readout" denote readout 1 and readout 2, i.e. the $\lambda/4$ readout resonators of Q_1 and Q_2 (the meanders `Bus1` and `Bus_to_Q2` terminated by their coupling tees `TQ1` and `TQ2`, Fig. 11); this naming is used throughout the remainder of this section.

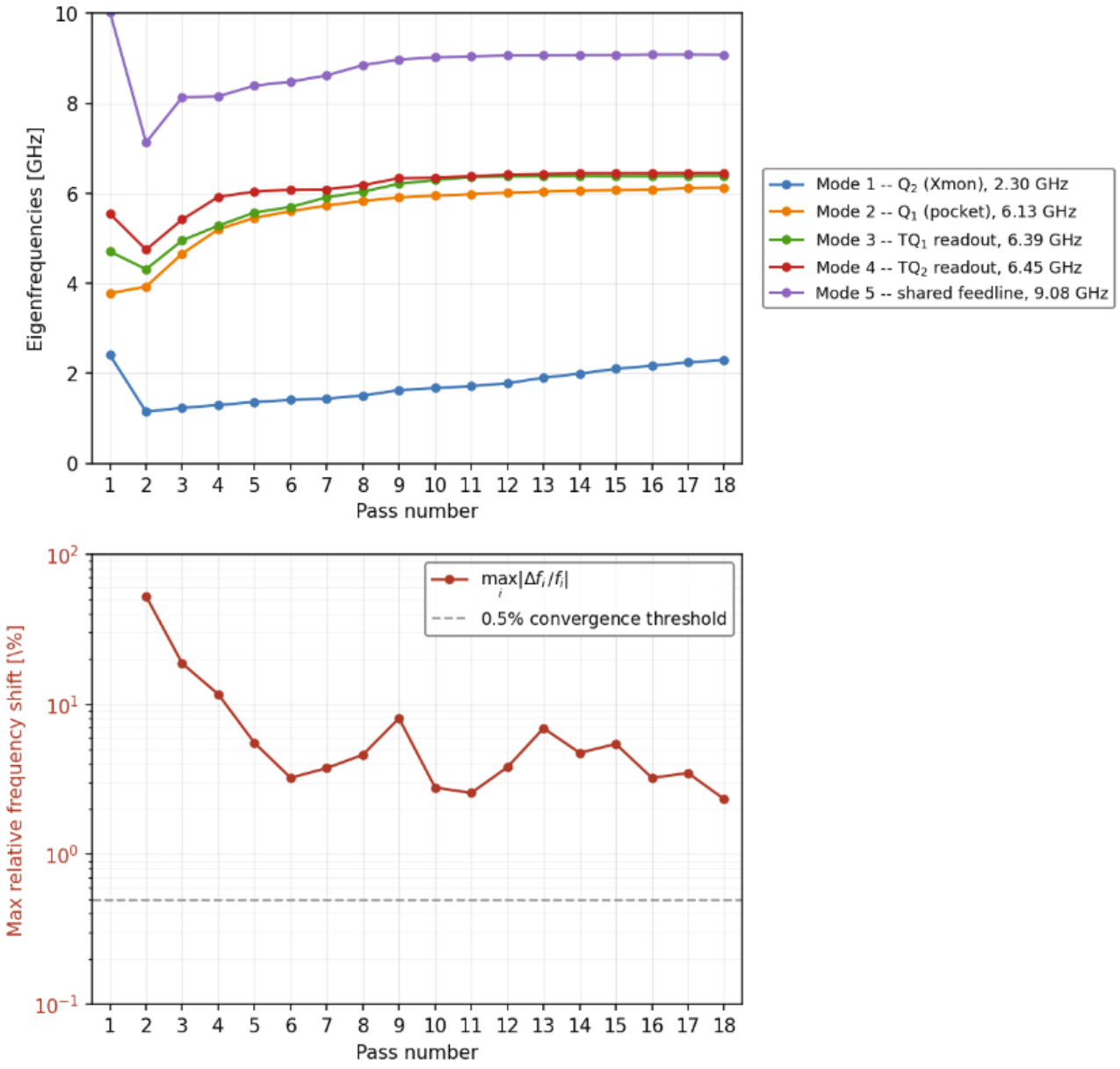


Figure 24: Convergence of the Model 2 full-chip HFSS eigenmode analysis. Same axes and generation procedure as Fig. 23. The convergence target was not meant within the range of passes **(bottom)**. due to Q_2's extended arm.

3.1.2 Qubit-mode E-field distributions

The pyEPR post-processing shows the electric field (E-field) visualisation (Figs. 25–28) where each pair combines a chip-scale overview with a zoom on the excited island. For both models, we focus on the Xmon and TransmonPocket E-field visualisation, and the remainder of the plots are left in Annex 5.2. pyEPR also extracts dielectric participation ratios, inserted in Tab. 4.

Model 1 (indirect coupling). Figure 25 shows the E-field of mode 2 which, by identification, corresponds to the `transmonPocketCL` object (Q1). The zoom on the plot helps notify a leakage

of the E-field through is meandered resonator `Bus1`, indicating a hybridised pair. This is confirmed by the junction participation ratio p_{22,Q_1} of 0.787 lower than the ideal junction participation of the `Xmon` whose value equals $p_{11,Q1} = 0.99$, indicating full spectral capture. Those values are reported in Table 4. Moreover, a residual value of $p_{33,Q1} = 0.205$ is captured by mode 3 confirms this spectral leakage.

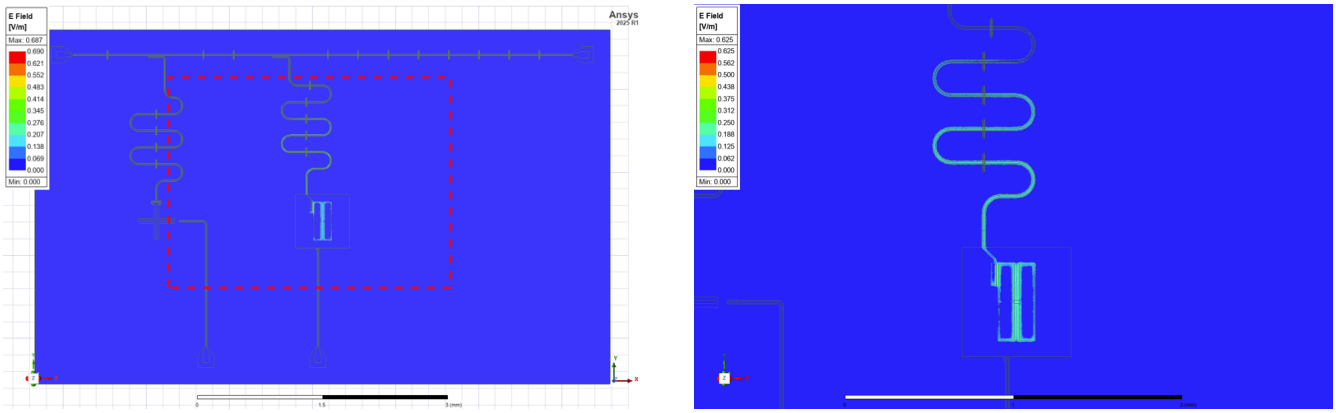


Figure 25: Q_1 eigenmode of Model 1 - indirect at 6.126 GHz ($p_{jj,Q_1} = 0.788$), hybridised with the readout. **(Left)**: full-chip overview with the pocket region outlined (red dashed box). **(Right)**: zoom on the pocket.

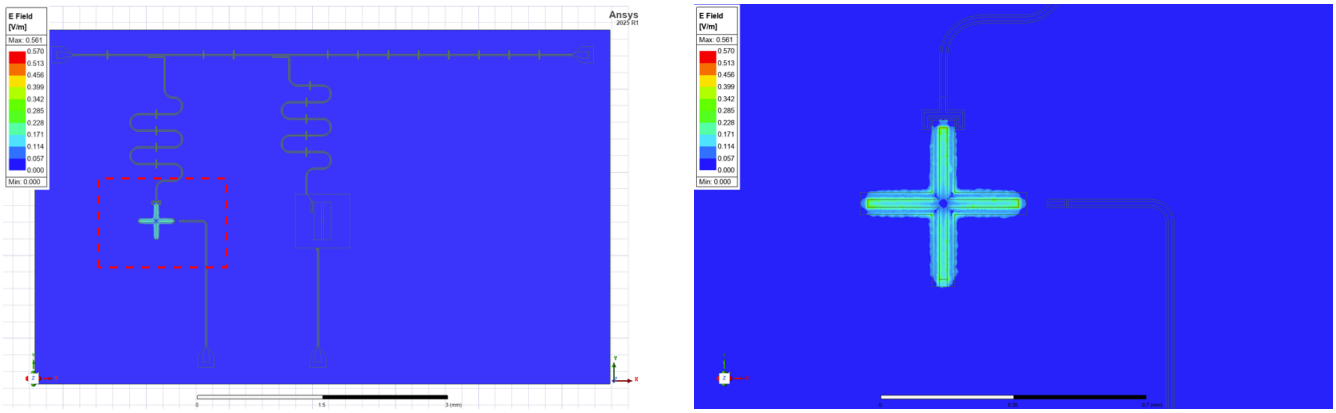


Figure 26: Q_2 eigenmode of Model 1 - indirect at 4.617 GHz ($p_{jj,Q_2} = 0.990$). **(Left)**: full-chip overview with the Xmon island outlined (red dashed box). **(Right)**: zoom on the Xmon island.

Model 2 (direct coupling) The same spectral leakage of model 1 is observed regarding the hybridisation of `transmonPocketCL` and its readout (Figure 27) with similar values of $p_{jj,Q1}$, reported in Table 4. However, a non-uniform E-field (Fig. 28) is observed, suggesting that the failure of the convergence criterion ($\max_m |\Delta f_m / f_m| < 0.5\%$) led to a wrong estimation of the `Xmon` qubit frequency, as the simulation stopped after the maximum number of passes defined by the simulation setup. This is analysed in Sec. 3.1.3.

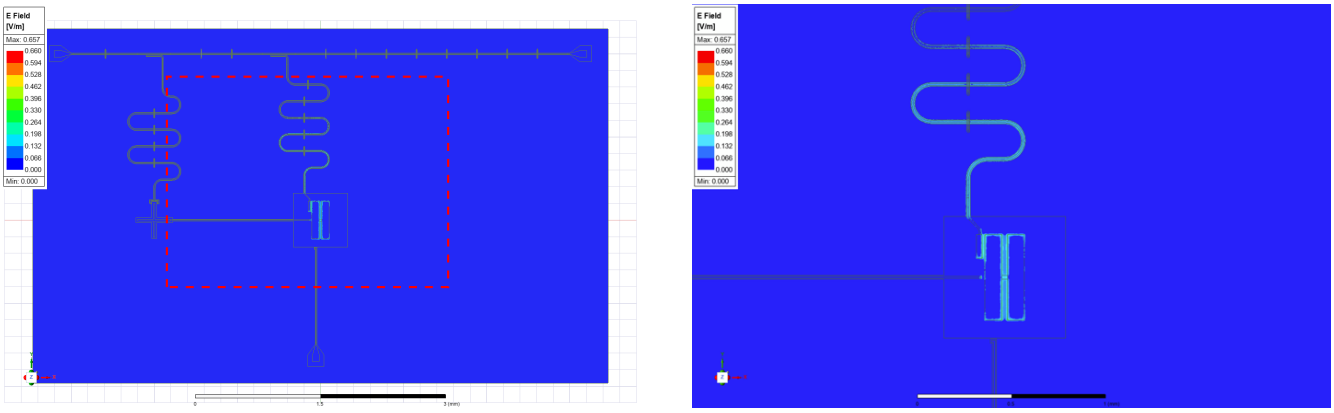


Figure 27: Q_1 eigenmode of Model 2 - direct at 6.126 GHz ($p_{jj,Q_1} = 0.778$). **(Left)**: full-chip overview with the pocket region outlined (red dashed box). **(Right)**: zoom on the pocket.

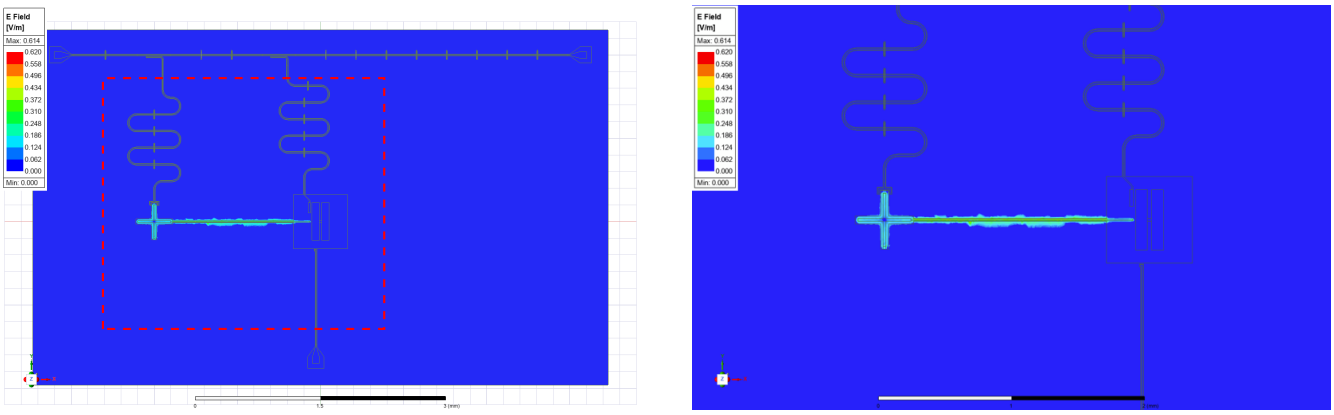


Figure 28: Q_2 eigenmode of Model 2 - direct at 2.295 GHz ($p_{jj,Q_2} = 0.974$). **(Left)**: full-chip overview with the Xmon island and coupler pad outlined (red dashed box). **(Right)**: zoom on the Xmon island and coupler pad.

3.1.3 Eigenmode identification and cross-model comparison

Table 4 summarises the pyEPR post-processing results for both models, together with the object identification of each mode. A row was added to compare the eigenfrequencies reported by Burignat; he returned neither p_{jj,Q_1} nor p_{diel}, as this section was left unfinished in his thesis. The eigenfrequencies are compared in Fig. 29: modes 2–5 are essentially indistinguishable between Model 1 and Model 2 (deviations below 15 MHz), whereas the deviations from Burignat's design are much more pronounced – most notably the readout 1 ($\sim$190 MHz drop) and the feedline mode ($\sim$2.1 GHz shift). Burignat's model, as a reminder, only included the `transmonPocketCL`, readout and `feedline` objects, no second qubit/readout. Figure 30a shows the substrate participation ratio $p_{\text{diel,main}}$ across both models, and Fig. 30b the junction participation ratios. The readout 1 eigenfrequency drop, the feedline shift, and the Q_2 frequency shift between the two models are thus discussed in Sec. 3.1.4.

	Model 1 (indirect)				Model 2 (direct)				Burignat [22]
Mode	f [GHz]	p_{jj,Q_1}	p_{jj,Q_2}	$p_{\text{diel,main}}$	f [GHz]	p_{jj,Q_1}	p_{jj,Q_2}	$p_{\text{diel,main}}$	f [GHz]
1	4.617	0	0.990	0.917	2.295	0	0.974	0.891	–
2	6.126	0.787	0	0.919	6.126	0.788	0	0.919	6.18
3	6.402	0.205	0	0.919	6.388	0.205	0	0.919	6.59
4	6.462	0	0	0.919	6.453	0	0	0.919	–
5	9.081	0	0	0.921	9.079	0	0	0.921	6.95

Mode identification. Mode 1: Q_2 (Xmon). Mode 2: Q_1 (pocket) Mode 3: TQ_1 readout Mode 4: TQ_2 readout. Mode 5: feedline.

Table 4: Converged HFSS eigenfrequencies f_m, junction participation ratios p_{jj,Q_i} and substrate dielectric participation ratios $p_{\text{diel,main}}$ of the five modes of Model 1 (indirect) and Model 2 (direct), with the corresponding frequencies of Burignat's reference design [22] shown for comparison (no dielectric participation reported). Below is the corresponding object identification to each mode.

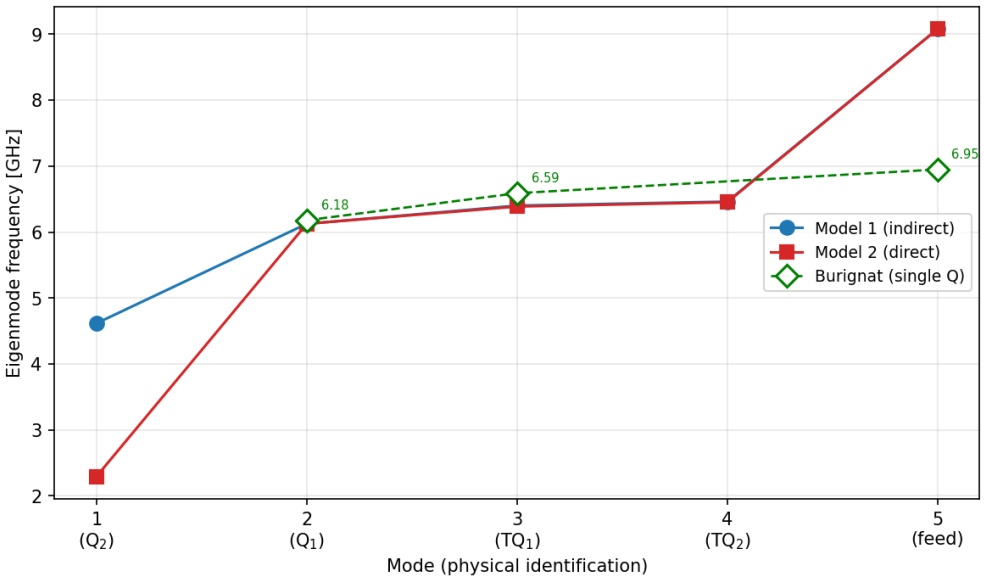
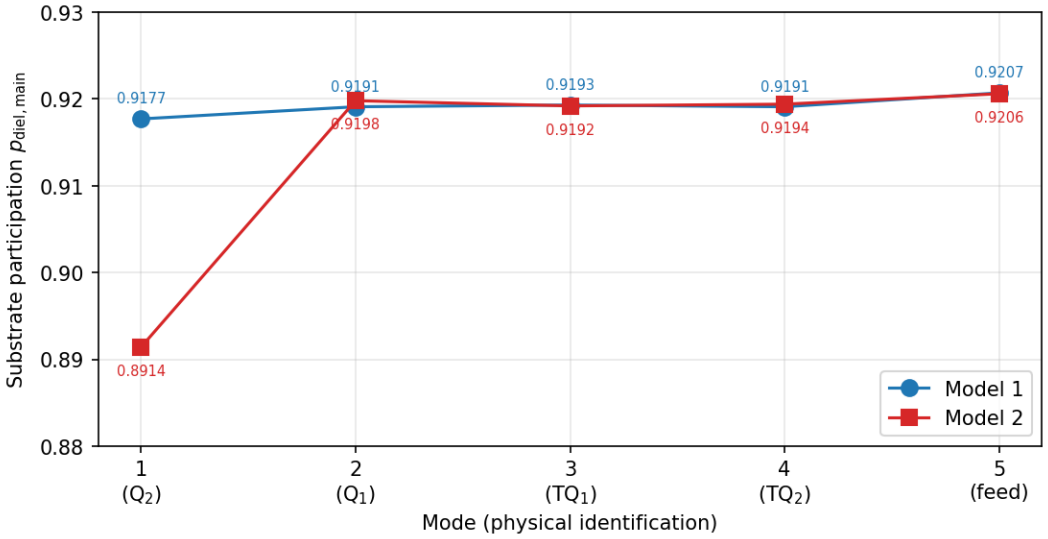
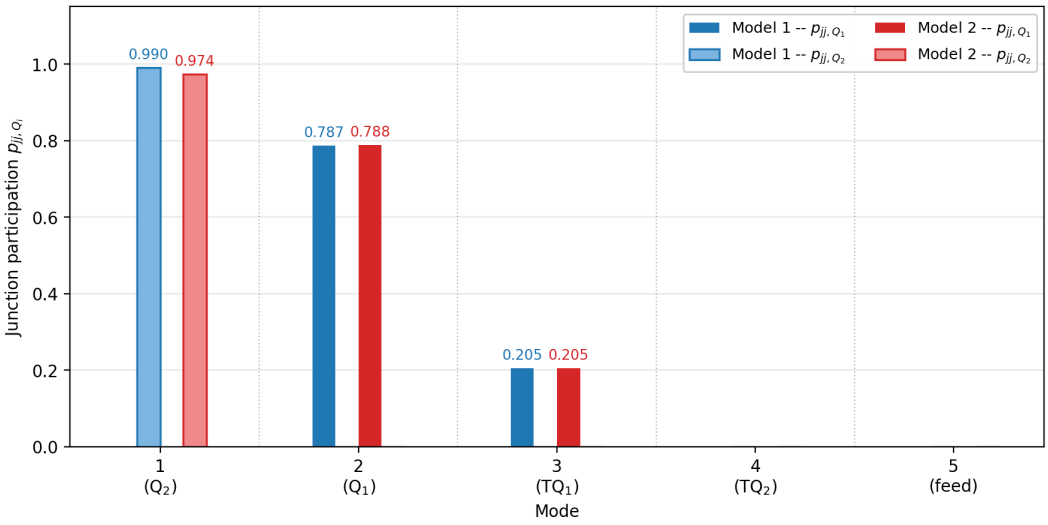


Figure 29: Eigenfrequencies f_m of the five HFSS modes for Model 1 (indirect coupling), Model 2 (direct coupling) and Burignat's reference design [22]. Modes 2–5 are essentially indistinguishable between the two models (deviations below 15 MHz), isolating the effect of the coupling scheme on mode 1 (Q_2). The differences with Burignat are much larger: TQ_1 drops by $\sim$190 MHz, reflecting the additional readout loading introduced by the ChargeLine, and the feedline mode shifts up by $\sim$2.1 GHz.



(a) Substrate dielectric participation ratios $p_{\text{diel,main}}$.



(b) Junction participation ratios p_{jj,Q_i} (dark: Q_1, light: Q_2).

Figure 30: Participation ratios of the five HFSS eigenmodes for Model 1 (indirect coupling, blue) and Model 2 (direct coupling, red), extracted from the pyEPR post-processing of Table 4. **(top)** Substrate dielectric participation $p_{\text{diel,main}}$: nearly flat across modes and models (~ 0.92), with only mode 1 of Model 2 dropping to 0.891. **(bottom)** Junction participation p_{jj,Q_i}: mode 1 is a pure Q_2 (Xmon) mode ($p_{jj,Q_2} \approx 1$), mode 2 is the hybridised Q_1/TQ_1 pair ($p_{jj,Q_1} \approx 0.79$ instead of unity), and the residual $p_{jj,Q_1} = 0.205$ on mode 3 (TQ_1) confirms the spectral leakage; modes 4 (TQ_2) and 5 (feedline) carry no junction energy. Both coupling schemes give near-identical junction participations, indicating that the direct/indirect choice does not affect the qubit–junction localisation.

3.1.4 Analysis

Readout 1 and its hybridisation with Q_1. Both meanders were drawn at the same 4.1 mm, the length returned by the analytical relation $l_r = c_0/(4 f_r \sqrt{\varepsilon_{\text{eff}}})$ of Sec. 1.3 [55] for a target close to 7 GHz. Readout 1 nevertheless comes out at 6.402 GHz, some 190 MHz below the value Burignat

obtained for the same nominal geometry (Fig. 29). The relation above treats the line as isolated, which it is not: a quarter-wave resonator is shorted at one end and open at the other, and its open end is a voltage antinode, so every capacitance seen there adds to the mode and pulls f_r down [56]. On this chip that end faces the qubit coupling pad and the coupled-line tee, and the full chip surrounds it with metal that Burignat's single-qubit design did not have: a second qubit, a second readout, the extra feedline segments and the ChargeLine routing.

The ChargeLine is the obvious suspect, being the main object added to the pocket, but it cannot be the direct cause. It attaches to the qubit island and not to the resonator, and Q_1 sits *below* readout 1, so the repulsion between the two pushes readout 1 up rather than down. Its influence on f_r can only be indirect. Settling the question would call for the capacitance matrix, and this is precisely where the full-chip cell gives nothing: the readout net is fused with the ground plane (Sec. 3.2.2), so neither the coupling capacitance to the qubit nor the strays towards the neighbouring conductors can be read from it. The cause therefore remains open. What can still be measured is the difference between the two readouts, which are drawn with the same length and the same coupling section: mode 4 carries no junction participation, so its 6.462 GHz is already a bare frequency, and it sits 117 MHz above the bare 6.345 GHz of `TQ1`, the only difference between the two being the qubit that hangs at the open end.

Model 2 non-convergence. That the Q_2 frequency drops between the two models is expected, and is not in itself a symptom of anything: the extended arm adds metal facing the ground plane and the pocket, hence capacitance on the Xmon island, and $\omega_q \simeq \sqrt{8E_J E_C}/\hbar$ with $E_C = e^2/2C_\Sigma$ falls accordingly. How much capacitance the arm actually adds is extracted in Sec. 3.2.2. What cannot be trusted is the value returned here: the 2.295 GHz of Table 4 comes from a solve that never met its convergence criterion and whose mode 1 was still rising at the last pass, and the same reservation applies to the anomalous $p_{\text{diel,main}} = 0.891$ (against ~ 0.919 for every other mode). Both numbers are reported below as evidence of the meshing problem, not as results.

Plotting the mesh (Fig. 31) helps identify the weaknesses of the HFSS simulation and of the *Qiskit Metal* layout. At each iteration, the algorithm refines the top 30 % of tetrahedra ranked by their local energy-error indicator, and stops once the eigenfrequency variation between two consecutive passes (Δf) falls below 0.5 %. The issue may arise from the fact that the JJ is modelled as a lumped element across a sub-µm gap: the field gradient is singular there, so the error indicator explodes and the refinement is monopolised by the junction (Fig. 31, bottom-right). The extended arm, which the E-field map shows to carry the strongest field of the whole mode (Fig. 31, top), receives comparatively few refinements, so the electric energy stored in the surrounding silicon is under-integrated, which pulls both $p_{\text{diel,main}}$ and f_{Q_2} off.

The E-field map also points to a weakness of the layout itself. Around the four regular arms of the cross, the ground plane is set back by the wide pocket cut-out, and the field spreads into the substrate over a comparable distance (Fig. 31 **(bottom left)**). Along the extended arm, the ground plane comes much closer: the same voltage now drops across a narrow gap, so the field is squeezed into a thin and intense band (Fig. 31, **(top)**). Its gradient must therefore be resolved over a few micrometres instead of a few tens, which is precisely what the coarse mesh there fails to do. Widening that cut-out to the value used on the other three arms would let the field spread out again and relax the gradients the mesher has to follow.

The simplest solution is to raise the number of passes from 18 to ≈ 25–30 (the typical range for pyEPR workflows on Xmon designs), hoping the algorithm eventually reaches the extended arm. One could also seed the mesh by specifying a maximum element length on the arm at the beginning of the simulation, forcing tetrahedra there regardless of the junction's dominance. Combining these should restore $p_{\text{diel,main}}$ to the $\sim$0.919 baseline of the other modes and pull f_{Q_2} back into agreement with Model 1.

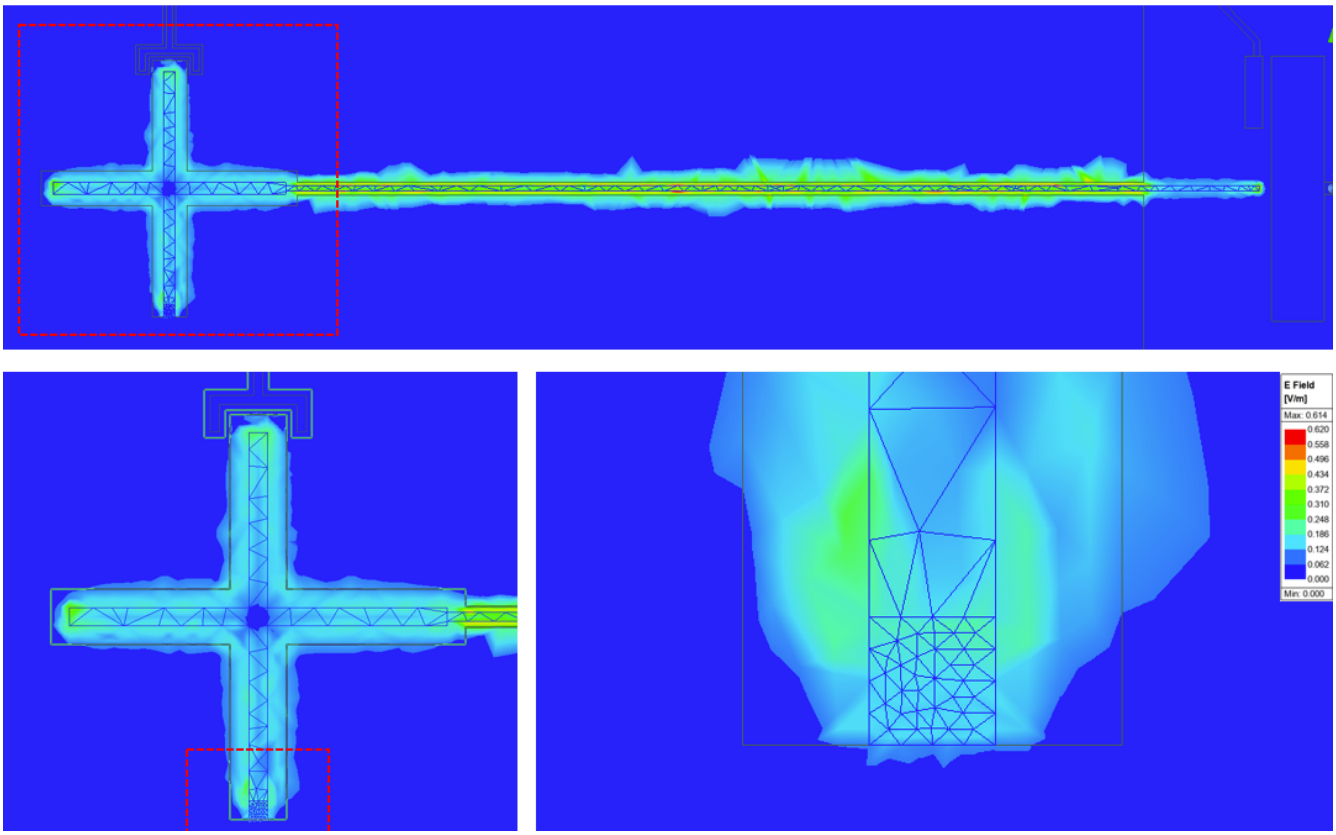


Figure 31: Adaptive HFSS mesh of Model 2 overlaid on the E-field map of mode 1 (Q_2), zoomed in progressively. **(Top)**: full-chip overview – the refinement is concentrated on the Xmon and its junction, while the extended arm and the feedline (both carrying non-negligible field) remain coarsely meshed. The red dashed box marks the region zoomed in the bottom-left panel. **(Bottom left)**: zoom on the Xmon cross, showing that the mesh density is biased towards the junction region rather than distributed over the four arms; the red dashed box marks the JJ region zoomed in the bottom-right panel. **(Bottom right)**: close-up on the Josephson junction, where the singular field gradient forces the adaptive algorithm to concentrate most of the refinement – at the expense of the extended arm.

3.2 Quasi-static 3D extractor (Q3D) and lumped oscillator model (LOM) simulation

Q3D is a quasi-static finite-element solver that computes the Maxwell capacitance matrix of a 3D multi-conductor structure embedded in a dielectric environment [57]. Given a set of N conductors, it solves Laplace's equation ($\nabla \cdot (\varepsilon \nabla \varphi) = 0$, where φ is the electrostatic potential and ε the permittivity of the medium) in the dielectric volume separating them (silicon substrate and vacuum, both free of volume charge) N times, each time holding one conductor at unit potential and all others grounded, and integrates the induced surface charge on every conductor to obtain the $N \times N$ Maxwell matrix

$$C_{ij} = -\left.\frac{\partial Q_i}{\partial V_j}\right|_{V_{k\neq j}=0}, \tag{23}$$

whose diagonal entries $C_{ii} = \sum_{j\neq i} |C_{ij}| + C_{i,\mathrm{gnd}}$ give the total capacitance of conductor i to all others plus ground, and whose off-diagonals C_{ij} $(i \neq j)$ are minus the mutual capacitance between conductors i and j. Two prerequisites shape how the result must be read: Q3D treats every galvanically continuous piece of metal as a single *net*, so a conductor made of several joined sub-features (*e.g.* a connector pad continuous with a meander line and a coupling bus) appears as one row/column of the matrix; and the solve is electrostatic (no time-dependent variables in Laplace's equation), so the returned C_{ij} are frequency-independent and reflect only the geometry and dielectric stack. As in the HFSS solve, the tetrahedral mesh is refined iteratively, densifying in regions of large field gradient until every matrix entry changes by less than the target 0.5 % between two successive passes, after which the extracted matrix is passed to the LOM analysis of Sec. 3.2.3 to compute the qubit-level parameters E_C, E_J/E_C, α and the dispersive shifts.

It is worth mentioning that, unlike the HFSS simulation which models the JJ as a linear $L_j \parallel C_j$, Q3D solves an electrostatic problem and thus is unable to model the inductance L_j of the JJ. It is just left as a gap, thus Q3D sees the `transmonPocketCL` as two parallel plates `pad_top_Q1` and `pad_bot_Q1` forming two distinct nets.

The Q3D solver is run with the settings summarised in Table 5: a minimum of 10 adaptive passes, a hard ceiling of 20 passes, and a per-pass error target of 0.5% on every matrix entry. The same setup is applied to both Model 1 and Model 2 to keep the two extractions on a common footing.

Parameter	Value
Minimum adaptive passes	10
Maximum adaptive passes	20
Per-pass error target	0.5%

Table 5: Q3D capacitance-extraction settings used for both Model 1 and Model 2.

3.2.1 Full-chip cell versus per-qubit cells

Burignat's Q3D workflow [22] extracts the capacitance matrix from a minimal cell: a bounding box that contains only the transmon pocket, its junction gap and a truncated stub of the readout

meander (Fig. 32).

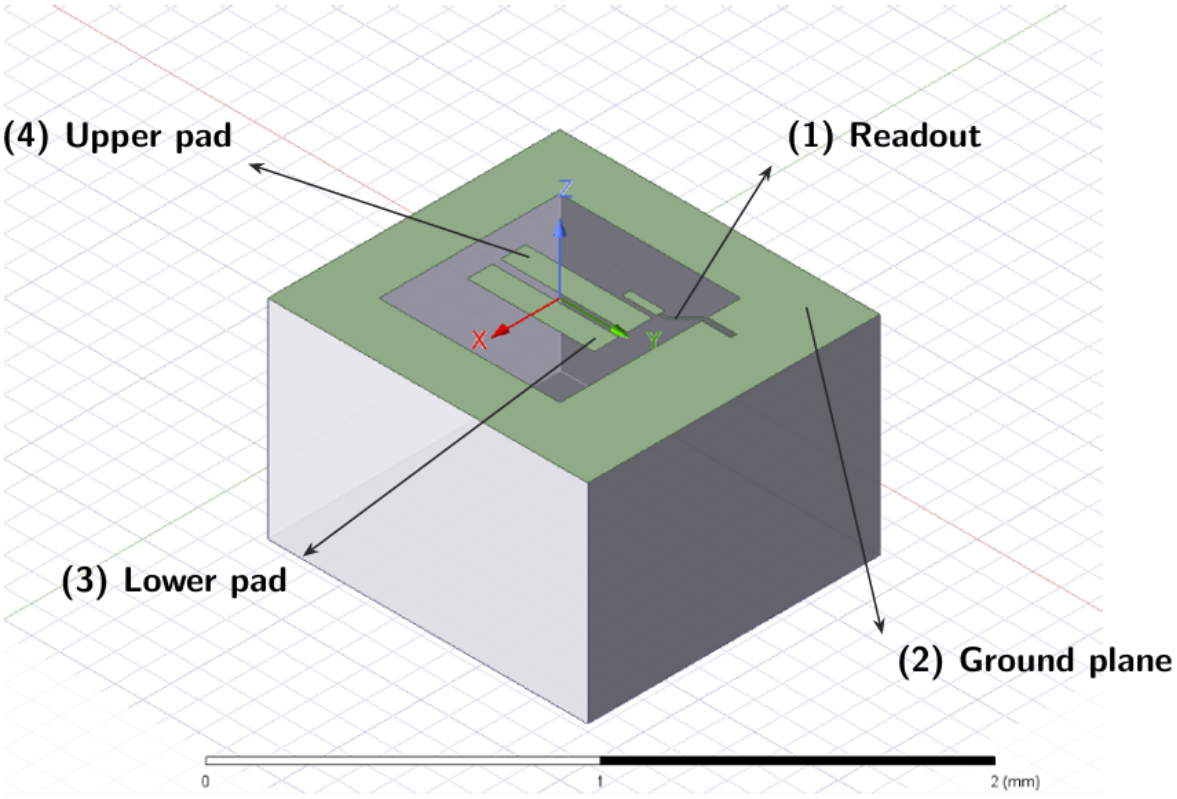


Figure 32: Burignat's Q3D cell model from Ansys Q3D, showing nodal configuration. The axis is centered at the JJ. **Node 1**: readout connector. **Node 2:** ground plane. **Node 3:** lower pad. **Node 4:** upper pad.

For the two-qubit topologies of this thesis, however, three coupling paths must be captured simultaneously to reconstruct the correct Hamiltonian: the qubit–qubit coupling C_{12} (direct or indirect depending on the model), the qubit–readout coupling $C_{q_i r_i}$ of each pair, and the residual coupling of every net to the shared feedline. Splitting the layout into per-qubit cells would require stitching these mutuals together *a posteriori* from overlapping extractions, with no guarantee of charge conservation and no faithful representation of the shared ground plane. We therefore render the full chip inside a single Q3D cell (Figures 33, 34), so that every pairwise capacitance appears explicitly as one entry of the capacitance matrices of Figures 35, 36.

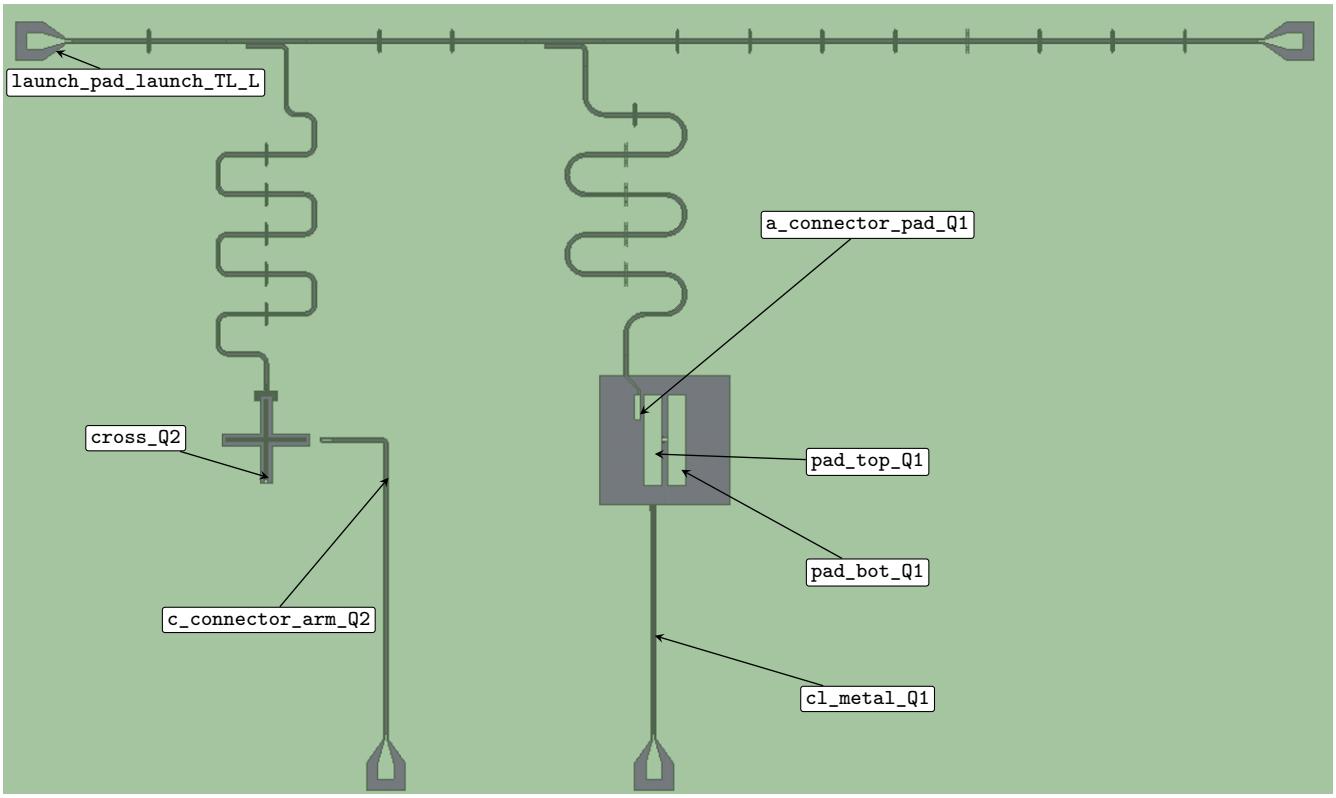


Figure 33: Q3D rendering of the indirect-coupling layout (Model 1), with each conductor of the extracted capacitance matrix (Fig. 35) annotated by its net name. The two qubit pads of Q_1 (`pad_top_Q1`, `pad_bot_Q1`) appear as distinct nets across the junction gap; the Xmon island (`cross_Q2`) and its charge line (`c_connector_arm_Q2`) are separate galvanic pieces; the Q_1 readout net (`a_connector_pad_Q1`) includes the connector pad, the meander `Bus1` and the coupled line of `TQ1` as one continuous conductor; and the feedline (`launch_pad_launch_TL_L`) spans launcher, feedline and second launcher.

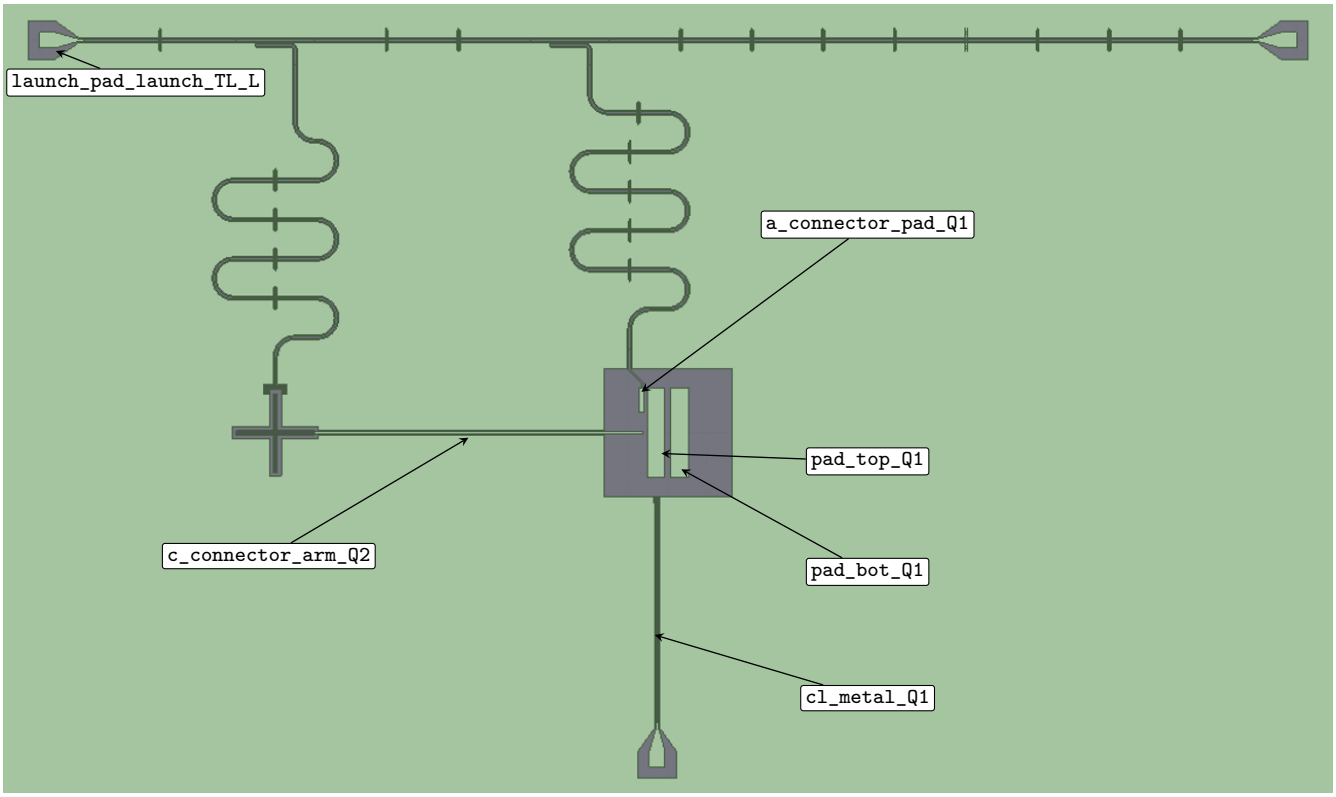


Figure 34: Q3D rendering of the direct-coupling layout (Model 2), same annotation convention as Fig. 33. The Xmon island and its coupler arm are now a single net (`c_connector_arm_Q2`) — `cross_Q2` has disappeared through the galvanic extension of the coupler pad that approaches Q_1, forming the small capacitive gap responsible for the direct qubit–qubit coupling C_{12}.

3.2.2 Capacitance matrix extraction

This section runs the Q3D simulation on both models. Both extractions reached the 0.5% target before the maximum pass count, so the returned matrices are treated as converged (16 passes for the indirect model and 17 for the direct); This yields the capacitance matrix heatmap shown in Figures 35 and 36.

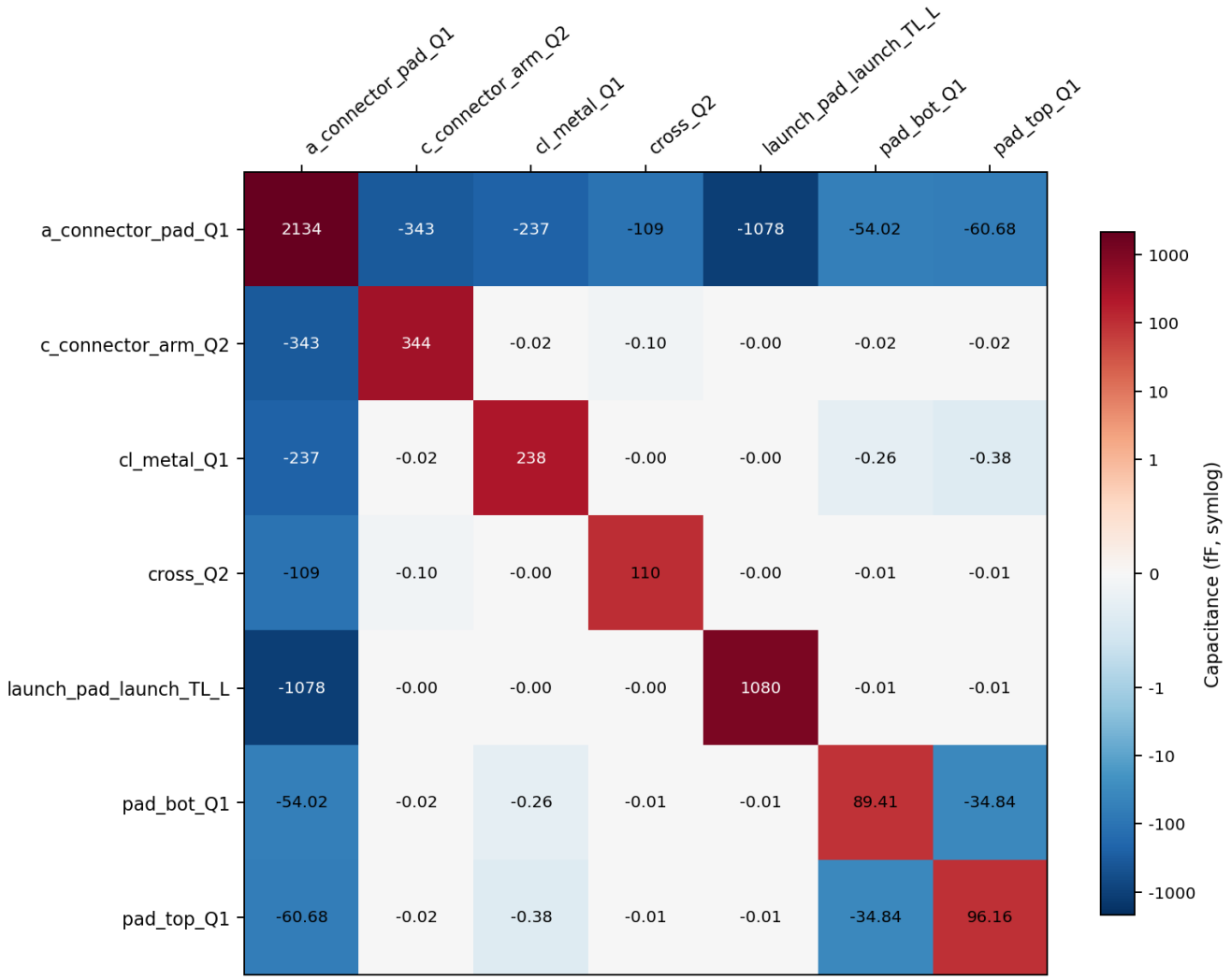


Figure 35: Q3D-extracted Maxwell capacitance matrix of the indirect-coupling layout (Model 1), values in fF. Diagonal entries give the total capacitance of each net to all others plus ground; off-diagonal entries are minus the pairwise mutual capacitance. Colour scale is symmetric-logarithmic to keep the sub-fF mutuals visible alongside the pF-scale diagonals of the composite readout and feedline nets.

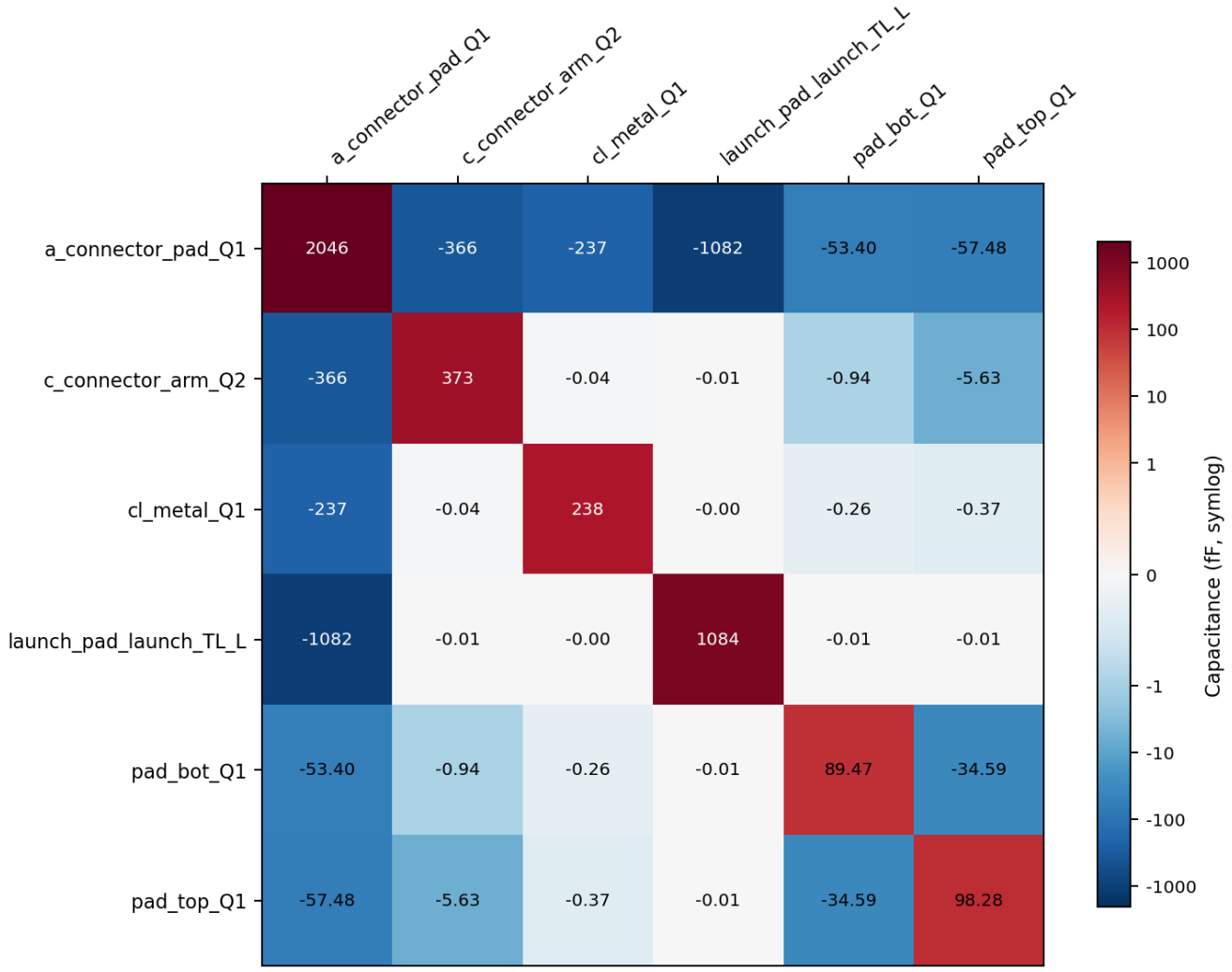


Figure 36: Q3D-extracted Maxwell capacitance matrix of the direct-coupling layout (Model 2), values in fF. Same conventions as Fig. 35; `cross_Q2` has merged into `c_connector_arm_Q2` through the coupling-pad extension, leaving a 6x6 matrix.

A first reading of Figs. 35 and 36 shows seven and six nets, whereas fourteen and twelve components were rendered. This is not a rendering failure but the net-merging rule of Sec. 3.2 at work. One merge was designed in: in Model 2 the Xmon island and its coupler arm are galvanically continuous, so they share a single row. The other was not expected: each readout resonator is a quarter-wave line, hence shorted to the ground plane at one end by construction. Ground plane, both resonators and the coupling tees therefore form one single conductor, which Q3D names after the first metal piece it renders: `a_connector_pad_Q1`, the connector pad of Q_1 (first entry of the capacitance matrix). A label that says nothing about what the net really is.

The extracted values leave no doubt about the identity of that net. Its self-capacitance reaches 2 134 fF in Model 1 and 2 046 fF in Model 2, one to two orders of magnitude above every qubit net and far beyond what a sub-millimetre pad could carry. It is also the dominant capacitive partner of every other conductor: once its contribution is subtracted, each remaining net retains between 0.08 and 0.3 fF, i.e. numerically nothing to screen against. That is exactly what a ground plane does: every conductor on the chip terminates its field lines on it. The diagonal entry must then

be read for what it is. Following the definition $C_{ii} = \sum_{j \neq i} |C_{ij}| + C_{i,\mathrm{gnd}}$ given above, the 2 134 fF of Model 1 split into 1 882 fF of mutual capacitance towards the other six nets and only 252 fF towards the reference (1 796 and 250 fF in Model 2). It is therefore not the self-capacitance of a connector pad, but the total capacitance of a very large conductor: the ground plane together with both readout resonators facing every other net of the chip.

Two consequences follow. First, this net is the reference of the circuit: its row and column are removed from the matrix before the LOM quantisation of Sec. 3.2.3, so that the qubit potentials are counted against the ground plane itself. Keeping it instead overestimates E_{C,Q_2} by 42 % (246 MHz against 173 MHz). Second, every capacitance internal to that net disappears with it. The qubit–readout capacitance C_g, and with it g_{qr} and the dispersive shift χ, is therefore out of reach of this cell and left to the eigenmode branch of Sec. 3.1 (implemented in Annex 5 but not processed); and so is the readout chain that links the two qubits of Model 1, whose g_{12} in Table 6 reduces to the residual stray pad-to-arm coupling alone. Only the first is a genuine loss: Model 1 carries no dedicated coupling bus (Sec. 2.1.6), so no interaction of the form of Eq. (12) is being missed: the readout chain mediates only at fourth order and through a matched 50 Ω continuum rather than a resonator mode, so the stray term is expected to dominate anyway. This is the price of the full-chip cell, which Burignat's minimal cell (Fig. 32) avoids only because his bounding box truncates the meander before its shorted end, so the short is never rendered.

The fix is cheap, since one galvanic contact per resonator causes the whole merge. That contact is the end of the coupled segment of `TQ1` (resp. `TQ2`) at which the $\lambda/4$ line runs into the ground plane: both tees are instantiated with `open_termination = False`, which is precisely what makes the line quarter-wave. It suffices to open that end in the rendering, by re-rendering the electrostatic cell with `open_termination = True` on the two tees; the renderer then leaves one CPW gap (6 µm) of etched ground around the end of the line. The ground plane, the connector pad and the meander are then seen as three separate conductors, and C_g, g_{qr} and χ become directly extractable. This concerns the electrostatic extraction only: the fabricated layout and the eigenmode solve keep the short that makes the line a quarter-wave resonator.

3.2.3 LOM analysis

The lumped oscillator model (LOM) represents the chip as a circuit graph in which every element (capacitors, inductors, Josephson junctions) is approximated by a discrete lumped element, so that the Hamiltonian parameters follow from linear algebra on the Maxwell capacitance matrix rather than from a spatial discretisation of Maxwell's equations [58]. Burignat runs it on the single-junction cell of Fig. 32, with the JJ parameters as $L_j = 10$ nH, $C_j = 2$ fF and the readout frequency `freq_readout` $= 7.0$ GHz as bare inputs; the output's values are reported in Table 6.

```
#LOM analysis

c2.setup.junctions=Dict(Lj=10, Cj=2)
c2.setup.freq_readout = 7.0
c2.setup.freq_bus = []

c2.run_lom()
c2.lumped_oscillator_all
```

Figure 37: Single-qubit LOM simulation of Burignat's design [22]. Values are reported in Table 6.

That routine cannot be reused on the layouts of this work. The cell of Sec. 3.2.1 contains two junctions, whereas `run_lom()` quantises a single one and therefore has no output for a qubit–qubit coupling g_{12}. We therefore bypass `run_lom()` and evaluate the LOM quantisation directly from the extracted Maxwell matrix, following the same formalism [58]. The full derivation is explained in Annex. 5.3 and the associated code is reproduced in Annex 5. The method returns E_C, E_J, E_J/E_C, ω_q, α for both qubits and the qubit–qubit coupling g_{12} (Tab. 6).

	Model 1 (indirect)		Model 2 (direct)		Burignat [22]
Quantity	Q_1	Q_2	Q_1	Q_2	Q
E_C [MHz]	294.772	173.478	293.226	51.766	260.2
E_J [GHz]	16.346	16.346	16.346	16.346	16.34
E_J/E_C	55.453	94.226	55.746	315.770	62.8
f_{lin} [GHz]	6.209	4.763	6.192	2.602	6.18
$\omega_q/2\pi$ [GHz]	5.914	4.589	5.899	2.550	5.56
$\alpha/2\pi$ [MHz]	−294.772	−173.478	−293.226	−51.766	−293.4
C_Σ [fF]	65.712	111.658	66.059	374.188	74.45
$g_{12}/2\pi$ [MHz]	0.034		25.890		–

Table 6: Hamiltonian parameters returned by the two-qubit LOM implementation of Eqs. (27)–(31), applied to the Q3D matrices of Figs. 35 and 36, with Burignat's single-qubit `run_lom()` output (Fig. 37) [22] shown for comparison. Same junction parameters throughout: $L_{j,1} = L_{j,2} = 10$ nH, $C_j = 2$ fF. $f_{\text{lin}} = \sqrt{8E_JE_C}/h = 1/2\pi\sqrt{L_jC_\Sigma}$ is the frequency the mode would have if the junction were reduced to its linear inductance L_j, directly comparable to the HFSS eigenfrequencies of Table 4; $\omega_q/2\pi = f_{\text{lin}} - E_C/h$ is the transmon $|0\rangle \to |1\rangle$ transition. Burignat's entry reports ω_q from `run_lom()` and f_{lin} from his HFSS eigenmode validation. The qubit–qubit coupling g_{12} is a property of the pair and is therefore reported once per model; Burignat's single-qubit design has no equivalent.

The table shows that the coupler acts almost entirely on Q_2. Extending the cross arm multiplies its island capacitance by more than three, $C_\Sigma = 111.7$ fF $\to 374.2$ fF, because the arm faces the ground plane along its whole length and its tip faces a pad of Q_1. Since $E_C = e^2/2C_\Sigma$ and

$f_{\mathrm{lin}} = \sqrt{8E_J E_C}/h \propto 1/\sqrt{C_\Sigma}$, the charging energy falls in the same proportion, from 173.5 to 51.8 MHz, and the frequency by $\sqrt{374.2/111.7} = 1.83$, from 4.763 to 2.602 GHz – the drop already observed in the eigenmode branch (Sec. 3.1.1). Two side effects come with it: the anharmonicity, equal to $-E_C$, collapses to -51.8 MHz, and E_J/E_C reaches 316, well above the transmon window, so a drive on Q_2 would leak heavily towards $|2\rangle$. Q_1 is left untouched, its C_Σ moving by 0.5 % only, which already answers part of the question raised in Sec. 1.3: the direct coupler perturbs the qubit that carries it, not its neighbour.

One consequence should be stated here. Taken alone, $g_{12}/2\pi = 25.89$ MHz would move an excitation from one qubit to the other in $\pi/2g_{12} \approx 10$ ns, but only if the two were resonant. They are not: Table 6 leaves them $\Delta_{12} \approx 3.35$ GHz apart, and the largest fraction of an excitation that can cross a detuned pair, $4g_{12}^2/(\Delta_{12}^2 + 4g_{12}^2)$, is then only 2×10^{-4} [20]. No exchange gate, iSWAP or $\sqrt{\mathrm{iSWAP}}$, can therefore be run on this layout as it stands, and what the coupler leaves is the dispersive ZZ interaction of Annex 5.1. The detuning being itself a consequence of the loading discussed above, the same geometric change corrects both.

3.3 Q3D and HFSS comparison

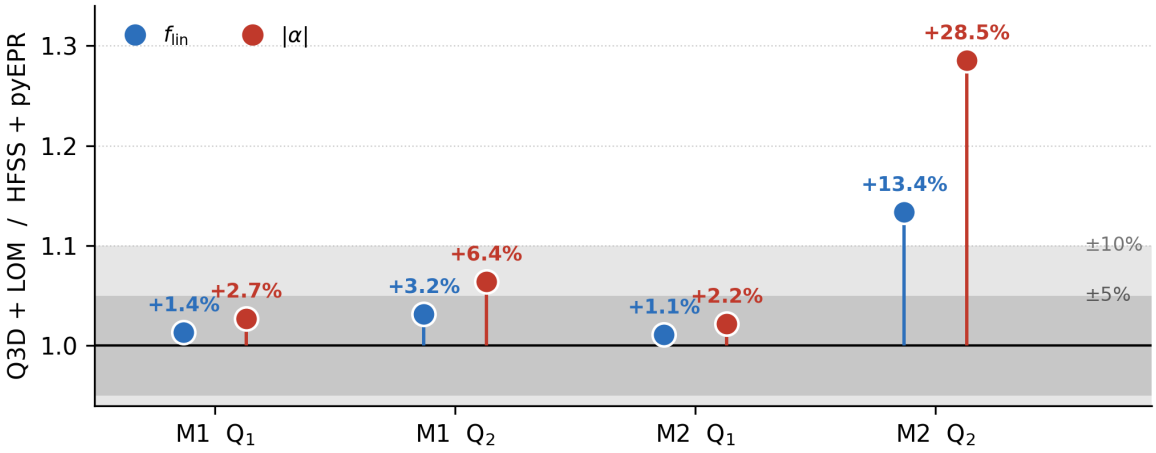


Figure 38: Ratio of the Q3D + LOM prediction to the HFSS + pyEPR result, for the two quantities both branches resolve: the linearised-junction frequency $f_{\mathrm{lin}} = \sqrt{8E_J E_C}/h$ (blue) and the mode anharmonicity $|\alpha| = p_{jj}^2 E_C$ (red). Shaded bands mark the ±5% and ±10% agreement envelopes. The three qubits whose adaptive mesh converged all fall within ±7%, which validates the two-qubit LOM implementation of Eqs. (27)–(30) against an independent full-wave solve. The Model 2 Q_2 mode is the sole outlier and its deviation quantifies the residual error left by the unconverged mesh of Sec. 3.1.1. All deviations are positive: the electrostatic branch systematically overestimates, as expected from an adaptive mesh that under-integrates the substrate field energy until convergence and therefore biases the eigenfrequencies downwards.

Two quantities are resolved by both branches and are plotted in Fig. 38. The first is the frequency the mode would have with a purely linear junction, $f_{\mathrm{lin}} = \sqrt{8E_J E_C}/h = 1/2\pi\sqrt{L_j C_\Sigma}$: HFSS returns it directly, since the junction is rendered there as a lumped $L_j \parallel C_j$, whereas Q3D never

models the junction at all – it is left as a gap, and L_j only enters at the quantisation step through E_J. The second is the anharmonicity, $|\alpha| = E_C$ for the LOM and $|\alpha| = p_{jj}^2 E_C$ after the EPR post-processing. Their ratio is plotted qubit by qubit, so unity means perfect agreement. The two branches share only the geometry and the junction inputs, and no parameter is fitted, so the comparison is a genuine cross-validation.

Three of the four qubits agree to 1.1%–3.2% on the frequency and 2.2%–6.4% on the anharmonicity, inside the $\pm 7\%$ envelope. All deviations are positive, the LOM always predicting the higher value, and their amplitude follows the qubit type rather than the coupling scheme: about 1% for the two pockets, 3.2% and 13.4% for the two Xmon.

The sole outlier is Q_2 of Model 2, at +13.4% and +28.5%, i.e. the mode whose mesh did not converge (Sec. 3.1.4). Its HFSS frequency was still rising at the last pass, hence underestimated, which is the sign needed for a positive ratio; the larger figure on $|\alpha| = p_{jj}^2 E_C$ follows from the squared dependence on p_{jj}. This point measures what the missing passes would still have to correct.

Two conclusions follow. For the three converged modes, two independent methods agree to better than 7% with no adjustable parameter, so the parameters of Tab. 6 can be used with confidence – including g_{12}, which only the LOM branch resolves. For Q_2 of Model 2, the Q3D + LOM value ($f_{\text{lin}} = 2.602$ GHz) is the more trustworthy of the two: it rests on an extraction that reached its target, its HFSS counterpart on the one mesh that did not.

4 Conclusion

This thesis addressed the design, simulation and characterisation of two coupled two-qubit superconducting architectures, developed in the continuity of Burignat's work [22]: an indirect scheme, in which a transmon pocket (Q_1) and an Xmon (Q_2) interact only through their $\lambda/4$ readout resonators and the shared feedline, and a direct scheme, obtained from the first by extending one Xmon arm into a coupling pad facing the Q_1 pocket. Both layouts were built in *Qiskit Metal*, several hard-coded geometric parameters of the stock components (`cl_stub_length`, `connector_arm_length`) being overridden at the layout level so that the rendered geometry matched the target specifications rather than the library defaults.

Both Ansys branches were repurposed relative to Burignat's usage. The EPR/HFSS analysis, which in his workflow served only as a modal validation step, was here used as a diagnostic tool: the dressed frequencies and participation ratios of the five modes exposed a convergence failure on Model 2 and traced it to an adaptive mesh monopolised by the junction singularity at the expense of the extended coupler arm. The Q3D extraction was in turn adapted from the per-qubit cell of Ref. [22] to a full-chip single-cell rendering, necessary to capture in one consistent Maxwell matrix the qubit–qubit coupling together with the qubit–readout, qubit–feedline and ChargeLine loading contributions, none of which survives independent per-qubit extractions without losing charge conservation. Since `run_lom()` quantises a single junction only, the LOM was re-implemented directly on the extracted matrix (Annex 5.3) and cross-validated against the eigenmode branch: three of the four qubit modes agree to better than 7 % on f_{lin} and $|\alpha|$ with no adjustable parameter.

This answers the question raised in Sec. 1.3. The direct coupler perturbs the qubit that carries it, and only that one. Q_1 is left untouched: its island capacitance moves by 0.5 % (65.7 → 66.1 fF) and its frequency by 15 MHz, so the pocket geometry validated in Ref. [22] survives the insertion of a neighbour and of a charge line. Q_2 does not: the extended arm multiplies C_Σ by 3.35 (111.7 → 374.2 fF), which collapses the charging energy from 173.5 to 51.8 MHz, the anharmonicity with it, and pushes E_J/E_C to 316, far outside the 50–100 transmon window, where charge dispersion is negligible but leakage towards $|2\rangle$ becomes prohibitive. A direct coupler can therefore not be dropped into an already validated single-qubit geometry: the qubit hosting it must be re-optimised, its neighbour need not. Two layout strategies follow directly from this asymmetry. The load can be shared rather than avoided, by replacing the arm galvanically attached to Q_2 with a floating pad terminating midway between the two qubits, so that each island sees roughly half of the added metal instead of one carrying all of it. Alternatively, the load can simply be reduced, by bringing the two qubits closer together: the same C_{12} is then reached with a much shorter arm, and ΔC_Σ scales with the length of metal that has been added, not with the coupling it produces.

The two coupling strengths, $g_{12}/2\pi \simeq 25.89$ MHz (direct) and 0.034 MHz (indirect), are of the expected order of magnitude but must be read differently. The direct value corresponds to a resonant iSWAP in $\pi/2g \approx 10$ ns, three orders of magnitude below the coherence times reported for this technology; it is, however, not directly usable as it stands, since the coupler itself detunes the pair by $\Delta_{12} \approx 3.3$ GHz, placing two fixed-frequency qubits deep in the dispersive regime where only the residual ZZ interaction (of the order of a few tens of kHz, Annex 5.1) remains. The indirect value calls for more care. As established in Sec. 2.1.6, Model 1 is not the canonical bus architecture of Sec. 1.1.8: the two qubits share no single mediating mode but a chain of three, so Eq. (12) does not transpose to it and the exchange survives only at fourth order, through a last link that

is a matched 50 Ω continuum rather than a resonator. The 0.034 MHz of Table 6 is accordingly not a bus-mediated coupling but the residual always-on floor of a chip on which no coupler has been drawn, dominated by stray capacitance between the two islands, and close to three orders of magnitude below the direct scheme, which is the figure that matters for isolation. Model 1 is thus better read as the uncoupled reference of the comparison than as an indirect coupler, and the architecture of Sec. 1.1.8 remains to be built.

Three limitations bound these results. The Model 2 eigenmode solve never met its 0.5 % target on the Xmon mode, so its 2.295 GHz is a lower bound and the Q3D value is the trustworthy one. The ground-plane fusion leaves C_g, g_{qr}, the dispersive shift χ and the Purcell-limited T_1 out of reach of the full-chip cell, so the part of the question concerning readout parameters remains open. Finally, the direct layout buys its coupling at the cost of Q_2's charge line, hence of independent drive on that qubit. Acceptable for a coupling study, not for an operational cell.

The natural continuation of this work is a third layout carrying a genuine bus coupler, i.e. the architecture of Sec. 1.1.8: a dedicated resonator, distinct from the two readout lines and detuned from both qubits, capacitively coupled to the Q_1 pocket at one end and to a free arm of the Q_2 cross at the other (Fig. 39). Such an element is inexpensive in *Qiskit Metal* (a third meander terminated by two coupling tees) and, decisively in view of the result above, it adds no metal to either qubit island: the capacitive load falls on the bus, not on the qubits, so neither E_J/E_C needs to be re-optimised. For qubit–bus couplings of the order of 100 MHz and detunings of 1 GHz to 3 GHz, Eq. (12) places the effective interaction in the few-MHz range, i.e. two-qubit gates in tens of nanoseconds. That sits between the 25.89 MHz of the direct pad, strong but bought at the cost of Q_2, and the residual floor of Model 1, which is unusable. Comparing that third model against the two studied here, on the same full-chip extraction workflow, would complete the coupler comparison this thesis opens. One could also go for a tunable coupler architecture (Sec. 1.1.9), but that is a redesign rather than an addition: it requires a SQUID and its flux-bias line, and the two qubits would have to be drawn again around the coupler.

Three shorter continuations follow from the limitations above. The first is a change to the rendering, not to the layout, and it concerns the two $\lambda/4$ readout resonators (`TQ1 + Bus1` for Q_1, `TQ2 + Bus_to_Q2` for Q_2), which the electrostatic cell returns fused with the ground plane into the single net labelled `a_connector_pad_Q1` in Fig. 33 and in the first row of Fig. 35. The extraction actually run (Annex 5) rendered both tees as they are drawn, that is with `open_termination = False`: the shorted end of each readout line was therefore rendered, and that single galvanic contact per resonator is what causes the merge. What remains to be run is the same extraction with `open_termination = True` on `TQ1` and `TQ2`, for the electrostatic pass only (Sec. 3.2): one CPW gap (6 µm) of ground metal then disappears at the end of each line, ground plane, connector pad and meander become three separate nets, and C_g becomes directly extractable, and with it g_{qr}, the dispersive shift χ and the Purcell-limited T_1. The fabricated layout and the eigenmode solve of Sec. 3.1 keep the short.

The second concerns the Model 2 solve itself. Seeding the mesh with a maximum element length on the coupler arm, and raising the pass ceiling to 25–30, should close the Model 2 convergence gap.

The third bears on the coupler geometry. The Q3D matrix also shows that, of the 262 fF the coupler arm adds to Q_2, only 5.63 fF actually face Q_1: the remaining 98 % is arm-to-ground loading that

buys no coupling at all. Shortening the arm while holding the tip gap fixed, i.e. moving the two qubits closer as suggested above, should therefore recover the transmon window and raise g_{12} rather than lower it, since $g_{12} \propto 1/\sqrt{C_{\Sigma,Q_2}}$ at fixed C_{12}. Mapping g_{12} and E_J/E_C against the tip gap and the arm length as two separate variables, in the spirit of the sensitivity maps of Ref. [22], would confirm it. Sweeping L_j costs nothing in this workflow, since the junction never enters the electrostatic solve and appears only at the quantisation step, and it is the knob that sets where Q_2 lands once C_{Σ,Q_2} has been reduced. Closing the 3.3 GHz gap should be its target, because the iSWAP is the gate a fixed capacitive coupler implements naturally and it requires the two qubits to be resonant. Since a fixed-frequency pair held on resonance would exchange continuously and could no longer be addressed separately, the cell becomes operable only once that exchange can be switched off.

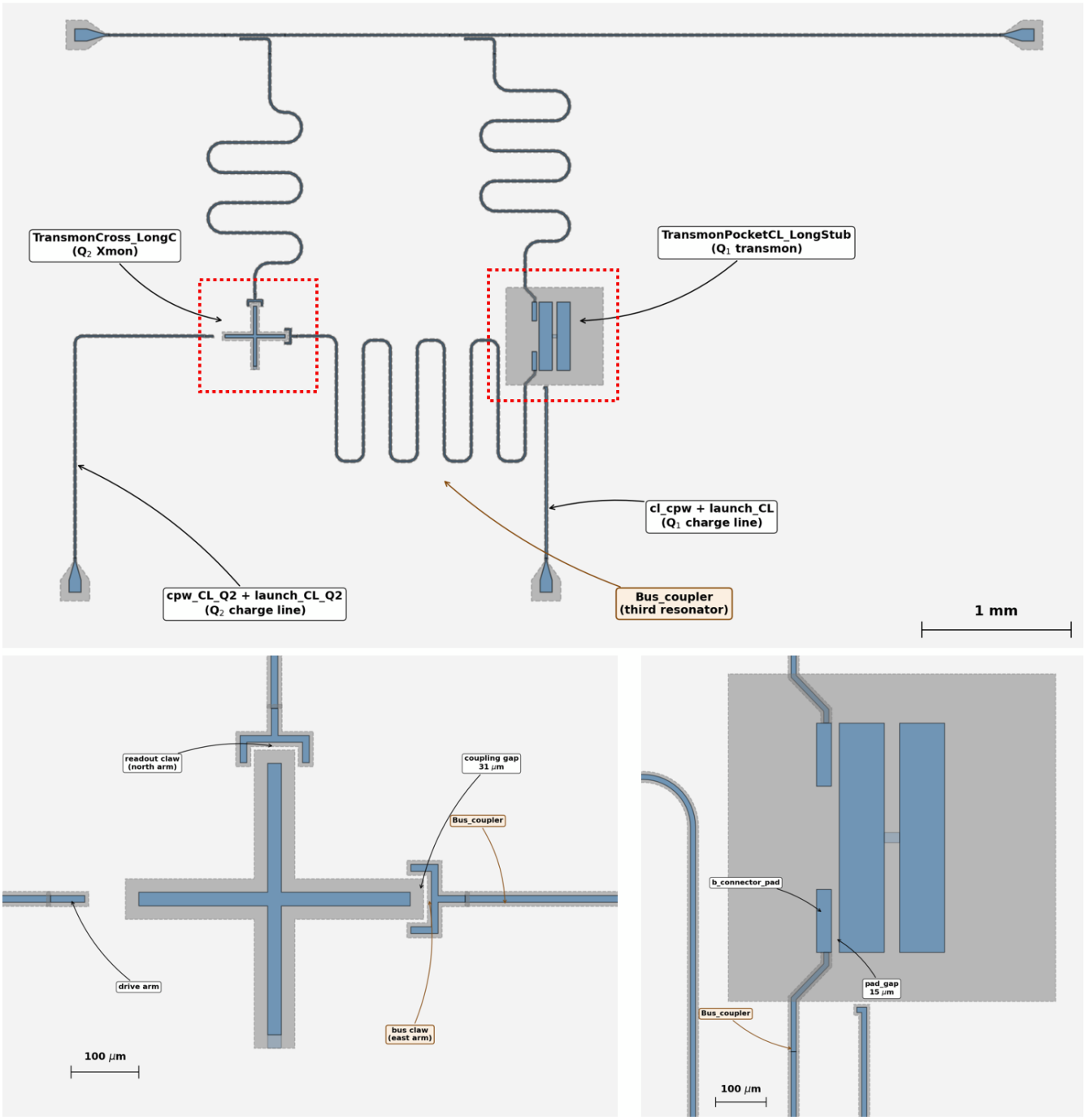


Figure 39: **(top)** Third proposed layout (Model 3), drawn in *Qiskit Metal* from the Model 1 chip. A dedicated $\lambda/2$ resonator (`Bus_coupler`, in orange; 7.2 mm, bare mode near 8 GHz) joins the east arm of the Xmon to the second connector pad of the Q_1 pocket. Both ends are capacitive, so no metal is added to either qubit island. **(bottom left)** Zoom on the Xmon, with the resonator claw on the Xmon east arm, metal-to-metal gap 31 μm; the drive line moves to the free west arm. **(bottom right)** Zoom on Q_1, with the resonator joining `b_connector_pad`, facing the `pad_top` island across `pad_gap` = 15 μm.

References

[1] J. M. Gambetta *et al.*, "Building logical qubits in a superconducting quantum computing system," *npj Quantum Information*, vol. 3, no. 1, p. 2, 2017. DOI: 10.1038/s41534-016-0004-0.

[2] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, IEEE, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[3] L. K. Grover, "Quantum Mechanics Helps in Searching for a Needle in a Haystack," *Physical Review Letters*, vol. 79, no. 2, pp. 325–328, 1997. DOI: 10.1103/PhysRevLett.79.325.

[4] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021. DOI: 10.22331/q-2021-04-15-433.

[5] J.-Z. Li *et al.*, "Proposal of realizing superadiabatic geometric quantum computation in decoherence-free subspaces," *Quantum Information Processing*, vol. 18, no. 1, p. 17, Nov. 2018. DOI: 10.1007/s11128-018-2134-0.

[6] A. G. Fowler *et al.*, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032 324, 2012. DOI: 10.1103/PhysRevA.86.032324.

[7] A. D. Córcoles *et al.*, "Demonstration of a quantum error detection code using a square lattice of four superconducting qubits," *Nat. Commun.*, vol. 6, p. 6979, 2015. DOI: 10.1038/ncomms7979.

[8] D. Ristè *et al.*, "Detecting bit-flip errors in a logical qubit using stabilizer measurements," *Nat. Commun.*, vol. 6, p. 6983, 2015. DOI: 10.1038/ncomms7983.

[9] J. Kelly *et al.*, "State preservation by repetitive error detection in a superconducting quantum circuit," *Nature*, vol. 519, p. 66, 2015. DOI: 10.1038/nature14270.

[10] M. H. Devoret and R. J. Schoelkopf, "Superconducting circuits for quantum information: An outlook," *Science*, vol. 339, p. 1169, 2013. DOI: 10.1126/science.1231930.

[11] T. P. Harty *et al.*, "High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit," *Phys. Rev. Lett.*, vol. 113, p. 220 501, 2014. DOI: 10.1103/PhysRevLett.113.220501.

[12] D. Nigg *et al.*, "Quantum computations on a topologically encoded qubit," *Science*, vol. 345, p. 302, 2014. DOI: 10.1126/science.1253742.

[13] C. Ospelkaus *et al.*, "Microwave quantum logic gates for trapped ions," *Nature*, vol. 476, p. 181, 2011. DOI: 10.1038/nature10290.

[14] M. D. Shulman *et al.*, "Demonstration of entanglement of electrostatically coupled singlet-triplet qubits," *Science*, vol. 336, p. 202, 2012. DOI: 10.1126/science.1217692.

[15] F. A. Zwanenburg *et al.*, "Silicon quantum electronics," *Rev. Mod. Phys.*, vol. 85, p. 961, 2013. DOI: 10.1103/RevModPhys.85.961.

[16] P. Gumann *et al.*, "Inductive measurement of optically hyperpolarized phosphorous donor nuclei in an isotopically enriched silicon-28 crystal," *Phys. Rev. Lett.*, vol. 113, p. 267 604, 2014. DOI: 10.1103/PhysRevLett.113.267604.

[17] D. Lu *et al.*, "Experimental estimation of average fidelity of a clifford gate on a 7-qubit quantum processor," *Phys. Rev. Lett.*, vol. 114, p. 140 505, 2015. DOI: 10.1103/PhysRevLett.114.140505.

[18] E. Martín-López *et al.*, "Experimental realization of shor's quantum factoring algorithm using qubit recycling," *Nat. Photon.*, vol. 6, p. 773, 2012. DOI: 10.1038/nphoton.2012.259.

[19] K. M. Maller *et al.*, "Rydberg-blockade controlled-not gate and entanglement in a two-dimensional array of neutral-atom qubits," *Phys. Rev. A*, vol. 92, p. 022 336, 2015. DOI: 10.1103/PhysRevA.92.022336.

[20] P. Krantz *et al.*, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021 318, 2019. DOI: 10.1063/1.5089550.

[21] M. Kjaergaard *et al.*, "Superconducting Qubits: Current State of Play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020, ISSN: 1947-5454. DOI: 10.1146/annurev-conmatphys-031119-050605.

[22] Y. Burignat, "A fast simulation-driven pre-fabrication workflow for transmon-based superconducting quantum chips," 2025. [Online]. Available: https://hdl.handle.net/2078.2/44632.

[23] M. Hatridge *et al.*, "Dispersive magnetometry with a quantum limited squid parametric amplifier," *Phys. Rev. B*, vol. 83, p. 134 501, 13 Apr. 2011. DOI: 10.1103/PhysRevB.83.134501.

[24] D. Gottesman, *Stabilizer codes and quantum error correction*, 1997. DOI: 10.48550/arXiv.quant-ph/9705052, pre-published.

[25] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018. DOI: 10.22331/q-2018-08-06-79.

[26] K. Alexander *et al.*, "A manufacturable platform for photonic quantum computing," *Nature*, vol. 641, no. 8064, pp. 876–883, 2025, ISSN: 1476-4687. DOI: 10.1038/s41586-025-08820-7.

[27] A. M. Steane, "Error correcting codes in quantum theory," *Phys. Rev. Lett.*, vol. 77, pp. 793–797, 5 Jul. 1996. DOI: 10.1103/PhysRevLett.77.793.

[28] E. Knill *et al.*, "Resilient quantum computation," *Science*, vol. 279, no. 5349, pp. 342–345, 1998. DOI: 10.1126/science.279.5349.342.

[29] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," *Physical Review A*, vol. 54, no. 2, pp. 1098–1105, 1996. DOI: 10.1103/PhysRevA.54.1098.

[30] A. M. Steane, "Multiple-Particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996. DOI: 10.1098/rspa.1996.0136.

[31] M. P. Bland *et al.*, "Millisecond lifetimes and coherence times in 2D transmon qubits," *Nature*, vol. 647, pp. 343–348, 2025. DOI: 10.1038/s41586-025-09687-4.

[32] D. Aharonov and M. Ben-Or, "Fault-tolerant quantum computation with constant error," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97, arXiv:quant-ph/9906129, New York, NY, USA: ACM, 1997, pp. 176–188. DOI: 10.1145/258533.258579.

[33] E. Knill *et al.*, "Resilient quantum computation: Error models and thresholds," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 365–384, 1998. DOI: 10.1098/rspa.1998.0166.

[34] J. Preskill, "Reliable quantum computers," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 385–410, 1998. DOI: 10.1098/rspa.1998.0167.

[35] D. Gottesman, "An introduction to quantum error correction and fault-tolerant quantum computation," in *Quantum Information Science and Its Contributions to Mathematics*, ser. Proceedings of Symposia in Applied Mathematics, S. J. Lomonaco, Ed., vol. 68, American Mathematical Society, 2010, pp. 13–58. DOI: 10.1090/psapm/068/2762145. eprint: 0904.2557.

[36] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1666-5.

[37] Y. Kim *et al.*, "Evidence for the utility of quantum computing before fault tolerance," *Nature*, vol. 618, no. 7965, pp. 500–505, 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-023-06096-3.

[38] R. Acharya *et al.*, "Quantum error correction below the surface code threshold," *Nature*, vol. 638, pp. 920–926, 2024, ISSN: 1476-4687. DOI: 10.1038/s41586-024-08449-y.

[39] R. Acharya *et al.*, "Suppressing quantum errors by scaling a surface code logical qubit," *Nature*, vol. 614, no. 7949, pp. 676–681, 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-022-05434-1.

[40] D. Bluvstein *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, vol. 626, no. 7997, pp. 58–65, 2024, ISSN: 1476-4687. DOI: 10.1038/s41586-023-06927-3.

[41] D. P. DiVincenzo, "The Physical Implementation of Quantum Computation," *Fortschritte der Physik*, vol. 48, no. 9–11, pp. 771–783, 2000, ISSN: 1521-3978. DOI: 10.1002/1521-3978(200009)48:9/11<771::AID-PROP771>3.0.CO;2-E.

[42] J. Koch *et al.*, "Charge-insensitive qubit design derived from the Cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042 319, 2007. DOI: 10.1103/PhysRevA.76.042319.

[43] S. Pettersson Fors *et al.*, *Comprehensive explanation of ZZ coupling in superconducting qubits*, 2024. DOI: 10.48550/arXiv.2408.15402. arXiv: 2408.15402 [quant-ph].

[44] J. Majer *et al.*, "Coupling superconducting qubits via a cavity bus," *Nature*, vol. 449, no. 7161, pp. 443–447, 2007, ISSN: 0028-0836. DOI: 10.1038/nature06184.

[45] F. Yan *et al.*, "Tunable coupling scheme for implementing high-fidelity two-qubit gates," *Phys. Rev. Appl.*, vol. 10, p. 054 062, 5 Nov. 2018. DOI: 10.1103/PhysRevApplied.10.054062.

[46] A. Blais *et al.*, "Circuit quantum electrodynamics," *Reviews of Modern Physics*, vol. 93, no. 2, p. 025 005, 2021. DOI: 10.1103/RevModPhys.93.025005.

[47] J. R. Schrieffer and P. A. Wolff, "Relation between the anderson and kondo hamiltonians," *Phys. Rev.*, vol. 149, pp. 491–492, 1966. DOI: 10.1103/PhysRev.149.491.

[48] J. Ruane *et al.*, "MIT quantum index: 2024 annual report," Massachusetts Institute of Technology, Initiative on the Digital Economy, Tech. Rep., 2025. [Online]. Available: https://qir.mit.edu/.

[49] IBM, *IBM makes quantum computing available on IBM Cloud to accelerate innovation*, Press release, 4 May 2016. [Online]. Available: https://newsroom.ibm.com/2016-05-04-IBM-Makes-Quantum-Computing-Available-on-IBM-Cloud-to-Accelerate-Innovation.

[50] IBM Quantum, *IBM quantum: Nighthawk processor and 2025 roadmap update*, Announced at the IBM Quantum Developer Conference 2025. [Online]. Available: https://www.ibm.com/quantum/blog.

[51] Google Quantum AI, *A preview of Bristlecone, Google's new quantum processor*, Google Research Blog, 5 March 2018. [Online]. Available: https://research.google/blog/a-preview-of-bristlecone-googles-new-quantum-processor/.

[52] Microsoft Azure Quantum *et al.*, "Interferometric single-shot parity measurement in InAs–Al hybrid devices," *Nature*, vol. 638, pp. 651–655, 2025, Accompanying the launch of the Majorana 1 topological quantum processor. DOI: 10.1038/s41586-024-08445-2.

[53] National Institute of Standards and Technology, "Post-quantum cryptography: FIPS 203, FIPS 204, and FIPS 205," U.S. Department of Commerce, National Institute of Standards and Technology, Tech. Rep., 2024, Published 13 August 2024. [Online]. Available: https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards.

[54] R. Barends *et al.*, "Coherent Josephson qubit suitable for scalable quantum integrated circuits," *Physical Review Letters*, vol. 111, no. 8, p. 080 502, 2013. DOI: 10.1103/PhysRevLett.111.080502.

[55] R. N. Simons, *Coplanar Waveguide Circuits, Components, and Systems*. John Wiley & Sons, 2004.

[56] I. Besedin and A. P. Menushenkov, "Quality factor of a transmission line coupled coplanar waveguide resonator," *EPJ Quantum Technology*, vol. 5, no. 1, p. 2, 2018, ISSN: 2196-0763. DOI: 10.1140/epjqt/s40507-018-0066-3.

[57] K. Nabors and J. White, "FastCap: A multipole accelerated 3-D capacitance extraction program," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 11, pp. 1447–1459, 1991. DOI: 10.1109/43.97624.

[58] Z. K. Minev *et al.*, *Circuit quantum electrodynamics (cQED) with modular quasi-lumped models*, 2021. DOI: 10.48550/arXiv.2103.10344.

5 Quantum processor implementation — Direct coupling model

```python
#!/usr/bin/env python
# coding: utf-8
"""
Quantum processor implementation -- DIRECT coupling model
Master thesis annex : qiskit-metal design + HFSS/Q3D/EPR simulation pipeline.
Two transmons (TransmonPocketCL Q1 + TransmonCross Q2) coupled directly via a
capacitive gap, each with its own CoupledLineTee readout on a shared feedline.
"""


# ============================================================================
# Annexe B -- Quantum Processor with Direct Coupler + Two-Tee Readout
```

```
# ============================================================================

# ============================================================================
# Environment Initialization
# ============================================================================

# Enable automatic reloading of modules when modifications are made
get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')

import qiskit_metal as metal
from qiskit_metal import designs, draw
from qiskit_metal import MetalGUI, Dict, open_docs, Headings

# Electromagnetic analysis toolkit
import pyEPR as epr

# Numerical / plotting
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Ansys renderer (for default options inspection)
from qiskit_metal.renderers.renderer_ansys.ansys_renderer import QAnsysRenderer
print('pyEPR :', repr(epr.__version__))

# ============================================================================
# Initialize Quantum Design Environment
# ============================================================================

# DesignPlanar : 2D circuit architecture for superconducting qubits
design = designs.DesignPlanar()

# Chip sizing -- extended along x to host Q1 + coupler + Q2 + readout subsystem
# Default DesignPlanar chip is ~9 x 6 mm. We explicitly set it for reproducibility.
design.chips.main.size.size_x = '8 mm'
design.chips.main.size.size_y = '6 mm'

# Enable design overwrite (iterative refinement without object collisions)
design.overwrite_enabled = True

# ============================================================================
# Quantum Component Library Import
# ============================================================================

# Qubits
# TransmonPocketCL : pocket-style transmon with built-in charge line (CL) stub
from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL

# TransmonCross : cross-shaped (Xmon) transmon with up to 3 connection pads
from qiskit_metal.qlibrary.qubits.transmon_cross import TransmonCross

# Waveguide routing
from qiskit_metal.qlibrary.tlines.straight_path import RouteStraight    # Direct CPW
from qiskit_metal.qlibrary.tlines.meandered import RouteMeander          # Meandered (resonator
    / bus)
```

```
from qiskit_metal.qlibrary.tlines.pathfinder import RoutePathfinder      # Auto-router fallback
from qiskit_metal.qlibrary.tlines.anchored_path import RouteAnchors      # Manual waypoints

# Couplers / tees
from qiskit_metal.qlibrary.couplers.coupled_line_tee import CoupledLineTee  # Capacitive tee (readout)
from qiskit_metal.qlibrary.couplers.line_tee         import LineTee         # Galvanic tee (Y-node)

# Terminations
from qiskit_metal.qlibrary.terminations.open_to_ground import OpenToGround
from qiskit_metal.qlibrary.terminations.short_to_ground import ShortToGround

# Launch pads (wirebond interface to chip carrier)
from qiskit_metal.qlibrary.terminations.launchpad_wb import LaunchpadWirebond

# ============================================================================
# Custom Component -- `TransmonPocketCL_LongStub`
# ============================================================================

# Subclass de TransmonPocketCL avec cl_cpw_length tunable
from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL
from qiskit_metal import draw, Dict


class TransmonPocketCL_LongStub(TransmonPocketCL):
    """TransmonPocketCL avec la longueur du cl_cpw exposée comme paramètre.

    Le composant parent hard-code la longueur de cl_cpw à `8 * cl_width`.
    Cette sous-classe expose un nouveau paramètre `cl_cpw_length` (par défaut
    '200 um' = ~2.5x la valeur d'origine de 80 µm). Permet d'allonger le stub
    interne sans toucher à `cl_width` et donc sans changer l'impédance.
    """

    default_options = Dict(
        **TransmonPocketCL.default_options,
        cl_cpw_length='200um',
    )

    def make_charge_line(self):
        """Override : utilise cl_cpw_length au lieu de 8*cl_width."""
        name = 'Charge_Line'
        p = self.p
        L = p.cl_cpw_length

        cl_arm = draw.box(0, 0, -p.cl_width, p.cl_length)
        cl_cpw = draw.box(0, 0, -L, p.cl_width)
        cl_metal = draw.unary_union([cl_arm, cl_cpw])
        cl_etcher = draw.buffer(cl_metal, p.cl_gap)

        port_line = draw.LineString([(-L, 0), (-L, p.cl_width)])

        polys = [cl_metal, cl_etcher, port_line]

        cl_rotate = 0
        if (abs(p.cl_pocket_edge) > 135) or (abs(p.cl_pocket_edge) < 45):
```

```
            polys = draw.translate(
                polys, -(p.pocket_width / 2 + p.cl_ground_gap + p.cl_gap),
                p.cl_off_center)
            if (abs(p.cl_pocket_edge) > 135):
                cl_rotate = 180
        else:
            polys = draw.translate(
                polys, -(p.pocket_height / 2 + p.cl_ground_gap + p.cl_gap),
                p.cl_off_center)
            cl_rotate = 90
            if (p.cl_pocket_edge < 0):
                cl_rotate = -90

        polys = draw.rotate(polys, p.orientation + cl_rotate, origin=(0, 0))
        polys = draw.translate(polys, p.pos_x, p.pos_y)
        [cl_metal, cl_etcher, port_line] = polys

        points = list(draw.shapely.geometry.shape(port_line).coords)
        self.add_pin(name, points, p.cl_width)
        self.add_qgeometry('poly', dict(cl_metal=cl_metal))
        self.add_qgeometry('poly', dict(cl_etcher=cl_etcher), subtract=True)

# ============================================================================
# Custom Component -- 'TransmonCross_LongC'
# ============================================================================

# Sous-classe de TransmonCross avec gap_extend_length exposé
from qiskit_metal.qlibrary.qubits.transmon_cross import TransmonCross


class TransmonCross_LongC(TransmonCross):
    """TransmonCross avec la longueur du bras type='1' (gap) exposée comme paramètre.

    Le composant parent hard-code la longueur du connector_arm de type '1' à
    '4 * claw_width' (= 40 µm avec claw_width=10 µm). Cette sous-classe expose
    un paramètre par-pad 'gap_extend_length'. Quand il vaut None (défaut), le
    comportement d'origine est conservé. Sinon, le bras métal du pad type-'1'
    est allongé à cette valeur -- permet de rapprocher la pointe du pad d'un
    autre composant pour former un direct coupler capacitif sans instancier
    une CPW externe.

    Convention littérature : ~20 µm de gap capacitif nu entre la pointe de
    l'extension et le métal voisin (typiquement l'etcher / pocket d'un pocket
    transmon).
    """

    # Ajoute gap_extend_length aux options par-pad (None → comportement parent)
    default_options = Dict(**TransmonCross.default_options)
    default_options['_default_connection_pads'] = Dict(
        **TransmonCross.default_options['_default_connection_pads'],
        gap_extend_length=None,  # None → 4*claw_width (comportement parent)
    )

    def make_connection_pad(self, name: str):
        """Override : bras type='1' allongeable via gap_extend_length."""
        p = self.p
```

```
        cross_width  = p.cross_width
        cross_length = p.cross_length
        cross_gap    = p.cross_gap
        chip = p.chip

        pc  = p.connection_pads[name]
        c_g = pc.claw_gap
        c_l = pc.claw_length
        c_w = pc.claw_width
        c_c_w = pc.claw_cpw_width
        c_c_l = pc.claw_cpw_length
        g_s   = pc.ground_spacing
        con_loc = pc.connector_location

        c_w_b = pc.claw_width_back if pc.claw_width_back is not None else c_w
        g_s_b = pc.ground_spacing_back if pc.ground_spacing_back is not None else g_s

        claw_cpw = draw.box(-c_w_b, -c_c_w/2, -c_c_l - c_w_b, c_c_w/2)

        if pc.connector_type == 0:
            # Claw connector : comportement parent inchangé
            t_claw_height = 2*c_g + 2*c_w + 2*g_s + 2*cross_gap + cross_width
            claw_base = draw.box(-c_w_b, -t_claw_height/2, c_l, t_claw_height/2)
            claw_subtract = draw.box(0, -t_claw_height/2 + c_w, c_l, t_claw_height/2 - c_w)
            claw_base = claw_base.difference(claw_subtract)
            connector_arm    = draw.shapely.ops.unary_union([claw_base, claw_cpw])
            connector_etcher = draw.buffer(connector_arm, c_g)
        else:
            # Gap connector (type='1') : longueur allongeable via gap_extend_length
            arm_len = pc.gap_extend_length if pc.gap_extend_length is not None else 4 * c_w
            connector_arm    = draw.box(0, -c_w/2, -arm_len, c_w/2)
            connector_etcher = draw.buffer(connector_arm, c_g)

        # port_line inchangé -- pin coté "arrière" du bras (côté cross)
        port_line = draw.LineString([
            (-c_c_l - c_w_b, -c_c_w/2),
            (-c_c_l - c_w_b,  c_c_w/2),
        ])

        claw_rotate = 0
        if con_loc > 135:
            claw_rotate = 180
        elif con_loc > 45:
            claw_rotate = -90

        polys = [connector_arm, connector_etcher, port_line]
        polys = draw.translate(polys, -(cross_length + cross_gap + g_s_b + c_g), 0)
        polys = draw.rotate(polys, claw_rotate, origin=(0, 0))
        polys = draw.rotate(polys, p.orientation, origin=(0, 0))
        polys = draw.translate(polys, p.pos_x, p.pos_y)
        [connector_arm, connector_etcher, port_line] = polys

        self.add_qgeometry("poly", {f"{name}_connector_arm": connector_arm}, chip=chip)
        self.add_qgeometry(
            "poly", {f"{name}_connector_etcher": connector_etcher},
            subtract=True, chip=chip,
```

```
        )
        self.add_pin(name, port_line.coords, c_c_w)

# ============================================================================
# A.1 -- Q1 : TransmonPocketCL with bias line *(charge line + readout pad + coupler pad)*
# ============================================================================

# Q1 -- pocket transmon with integrated charge line.
# Two connection pads on the pocket :
#   - 'a' : facing the readout resonator (north of Q1 after the 90° orientation)
#   - 'b' : facing Q2 (the Xmon, placed east of Q1) -- used for the direct coupler
#
# Orientation note : Q1 is rotated 90° in the world frame.
# - 'a' at connector_location='180' → routes toward TQ1 (north of Q1).
# - 'b' at connector_location='0'   → opposite side of the pocket, faces Q2.
# - cl_pocket_edge='0'              → charge line stub exits south (toward launch_CL).
import os
design.overwrite_enabled = True

Q1 = TransmonPocketCL_LongStub(design, 'Q1', options=dict(
    pos_x='0 mm',
    pos_y='0 mm',
    orientation='90',

    # Two connection pads -- pocket sides
    connection_pads=dict(
        a=dict(connector_location='180', claw_length='95 um'),  # → readout (TQ1)
        b=dict(connector_location='0',   claw_length='95 um'),  # → libre (sera atteint géomé
    triquement par le coupler polygonal)
    ),

    # CL stub geometry -- unchanged from the original design
    make_CL=True,            # Enable the integrated charge line stub
    cl_pocket_edge='0',      # CL stub side : east in local frame ⇒ south in world frame
    cl_off_center='50 um',   # Offset from pocket symmetry axis
    cl_width='10 um',        # CL CPW center-conductor width
    cl_gap='6 um',           # CL CPW gap to ground plane
    cl_length='20 um',       # CL stub protrusion into pocket
    cl_ground_gap='6 um',    # Gap at the open end of the stub
    cl_cpw_length='1.138 mm',  # stub long (=1.1 mm original + 38 µm) : la fin du stub tombe
    pile sur launch_CL.tie (y=-1.475 mm). Aucune CPW externe cpw_CL nécessaire ; l'etcher du
    pocket ne recouvre pas la CPW.
))

design.rebuild()

# ============================================================================
# A.2 -- Q2 : Xmon (TransmonCross), raised to y = 0.8 mm
# ============================================================================

design.overwrite_enabled = True

# Q2 -- Xmon avec pad C ALLONGÉ pour former le direct coupler capacitif Q2 ↔ Q1.
# Utilise TransmonCross_LongC (défini plus haut) qui expose gap_extend_length
# comme paramètre par-pad. Plus besoin d'instancier une CPW polygonale externe
# (FreeCoupler) -- le bras métal du pad c est étendu directement jusqu'à ~20 µm
```

```
# de Q1.
#
# Calcul du gap capacitif (convention littérature ~20 µm) :
#   • Base du bras pad c (world, après rotation/translation Q2)   : x = -1.720 mm
#   • Bord du pocket etcher ouest de Q1 (world, orientation=90°)  : x = -0.325 mm
#   • Pointe du bras à 20 µm du pocket etcher                     : x = -0.345 mm
#   • Longueur du bras : arm_len = -345 - (-1720) = 1375 µm       = 1.375 mm

Q2 = TransmonCross_LongC(design, 'Q2', options=dict(
    pos_x='-2 mm', pos_y='0 mm', orientation='0',
    cross_width='20 um', cross_length='200 um', cross_gap='20 um',
    connection_pads=dict(
        c=dict(
            connector_location='180',   # bras est (world) : face à Q1
            connector_type='1',         # gap-type (pas de claw)
            claw_length='30 um',
            claw_width='10 um',         # largeur du bras métal du direct coupler
            claw_gap='6 um',            # gap CPW de chaque côté du bras
            ground_spacing='-26 um',     # 20+54+6 = 80 µm entre arm tip et cross east
            gap_extend_length='1.674 mm',  # NOUVEAU : bras étendu jusqu'à ~20 µm du pocket Q1
        ),
        b=dict(connector_location='90', connector_type='0',
               claw_length='30 um', claw_width='10 um', claw_gap='6 um'),
    ),
))

design.rebuild()

# ============================================================================
# A.4 -- Two readout subsystems  *(TQ1+Bus1 for Q1, TQ2+Bus_to_Q2 for Q2)*
# ============================================================================

import numpy as np

# -- Monkey-patch the qiskit-metal 0.1.5 collision-check bug -----------------
# parse_value() does NOT convert 'true'/'false' strings to Python booleans, so
# the option advanced.avoid_collision='false' (the default!) is truthy and the
# collision check runs anyway, hitting the AttributeError on MultiPolygon
# components. We force-bypass it once per kernel session.
from qiskit_metal.qlibrary.tlines import anchored_path as _ap
def _always_unobstructed(self, segment):
    return True
_ap.RouteAnchors.unobstructed = _always_unobstructed
try:
    from qiskit_metal.qlibrary.tlines import meandered as _md
    _md.RouteMeander.unobstructed = _always_unobstructed
except (ImportError, AttributeError):
    pass
try:
    from qiskit_metal.qlibrary.tlines import pathfinder as _pf
    _pf.RoutePathfinder.unobstructed = _always_unobstructed
except (ImportError, AttributeError):
    pass
# ----------------------------------------------------------------------------

design.overwrite_enabled = True
```

```
343
344 # 1) TQ1 -- readout tee for Q1 (unchanged geometry)
345 TQ1 = CoupledLineTee(design, 'TQ1', options=dict(
346     pos_x='-0.5 mm',
347     pos_y='2 mm',
348     coupling_length='200 um',
349     open_termination=False,
350 ))
351
352 # 2) Bus1 -- readout meander Q1.a → TQ1.second_end
353 bus1 = RouteMeander(design, 'Bus1', options=dict(
354     total_length    = '4.1 mm',
355     hfss_wire_bonds = True,
356     fillet          = '99 um',
357     advanced=Dict(avoid_collision='false'),
358     pin_inputs=Dict(
359         start_pin=Dict(component='Q1',  pin='a'),
360         end_pin  =Dict(component='TQ1', pin='second_end'),
361     ),
362 ))
363
364 # 3) TQ2 -- the NEW readout tee for Q2 (same geometry as TQ1)
365 TQ2 = CoupledLineTee(design, 'TQ2', options=dict(
366     pos_x='-2 mm',
367     pos_y='2 mm',
368     coupling_length='200 um',
369     open_termination=False,
370 ))
371
372 # 4) Bus_to_Q2 -- readout meander Q2.c → TQ2.second_end
373 #    Q2.c is on the east arm of the Xmon at world (-1.9, 0.8) facing east.
374 #    TQ2.second_end is at world (-1.9, 1.875) facing south.
375 Bus_to_Q2 = RouteMeander(design, 'Bus_to_Q2', options=dict(
376     total_length    = '4.1 mm',
377     hfss_wire_bonds = True,
378     fillet          = '99 um',
379     advanced=Dict(avoid_collision='false'),
380     pin_inputs=Dict(
381         start_pin=Dict(component='Q2',  pin='b'),
382         end_pin  =Dict(component='TQ2', pin='second_end'),
383     ),
384 ))
385
386 design.rebuild()
387
388 # ============================================================================
389 # A.5 -- Control & readout interface  *(feedline through both tees, no separate drive line)*
390 # ============================================================================
391
392 # Launch pads -- interface to the chip carrier (wirebond pads).
393 design.overwrite_enabled = True
394
395 launch_TL_L = LaunchpadWirebond(design, 'launch_TL_L', options=dict(
396     pos_x='3 mm', pos_y='2 mm', orientation='180',
397 ))
398 launch_TL_R = LaunchpadWirebond(design, 'launch_TL_R', options=dict(
```

```
    pos_x='-3 mm', pos_y='2 mm',
))

launch_CL = LaunchpadWirebond(design, 'launch_CL', options=dict(
    pos_x='-0.055 mm',
    pos_y='-1.5 mm',
    orientation='90',
))

# Feedline east : launch_TL_L → TQ1.prime_end
cpw_eastmost = RouteStraight(design, 'cpw_eastmost', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='launch_TL_L', pin='tie'),
        end_pin  =Dict(component='TQ1',         pin='prime_end'),
    ),
))

# Feedline middle : TQ1.prime_start → TQ2.prime_end
cpw_middle = RouteStraight(design, 'cpw_middle', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='TQ1', pin='prime_start'),
        end_pin  =Dict(component='TQ2', pin='prime_end'),
    ),
))

# Feedline west : TQ2.prime_start → launch_TL_R
cpw_westmost = RouteStraight(design, 'cpw_westmost', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='TQ2',         pin='prime_start'),
        end_pin  =Dict(component='launch_TL_R', pin='tie'),
    ),
))

# Drive line Q1 -- pas de CPW externe.
# Le stub interne du Q1 (cl_cpw_length='1.138 mm') se termine exactement sur
# launch_CL.tie (y=-1.475 mm), donc les métaux se touchent et fusionnent
# naturellement dans le rendu GDS/HFSS. Aucun RouteStraight requis, et l'etcher
# du pocket ne peut pas recouvrir de CPW externe puisqu'il n'y en a pas.

# ============================================================================
# DIRECT COUPLER Q1 ↔ Q2 -- via l'extension du pad C du Xmon Q2
# ============================================================================
# Plus de FreeCoupler polygonal externe. Le bras métal du pad c de Q2 est
# allongé directement dans TransmonCross_LongC via gap_extend_length='1.375 mm'.
# Résultat : la pointe du bras s'arrête à ~20 µm de l'etcher du pocket de Q1
# → gap capacitif nu, convention littérature pour un direct coupler
# transmon-transmon. Le layout ne contient plus qu'UN seul objet pour le
# coupler (le bras du pad c lui-même), et Ansys voit deux nets propres
# (Q1 et Q2, dont le pad c allongé est parti intégrante).

design.rebuild()

# ============================================================================
```

```
# A.6 -- Final layout overview
# ============================================================================

# Full processor -- all components.
design.rebuild()
print('Components in design :')
for name, comp in design.components.items():
    print(f'  • {name:<20s} ({type(comp).__name__})')

# ============================================================================
# B -- Full-chip eigenmode simulation & EPR analysis
# ============================================================================

from qiskit_metal.analyses.quantization import EPRanalysis

# EPR analysis with the HFSS eigenmode solver on the full-chip design
eig_full = EPRanalysis(design, "hfss")
hfss     = eig_full.sim.renderer

# Launch Ansys Electronics Desktop and open a fresh eigenmode design container
hfss.start()
hfss.activate_ansys_design("FullChip_Eigenmode", 'eigenmode')

# Full chip render (Q1, Q2, both tees, both readout meanders, feedline segments,
# launchpads, Q1 CL stub). Le direct coupler est intégré dans le bras allongé du
# pad c de Q2 (voir TransmonCross_LongC + gap_extend_length) -- pas d'objet
# externe, pas d'open_pin.
hfss.render_design()

# Multi-mode setup
setup = hfss.pinfo.setup
setup.nmodes     = 5    # Q1 + Q2 + TQ1_readout + TQ2_readout + feedline
setup.n_modes    = 5
setup.passes     = 18
setup.min_passes = 8
setup.delta_f    = 0.5

# Josephson junctions -- Lj / Cj partagés (renomme en Lj1 / Lj2 pour un sweep)
pinfo = hfss.pinfo
pinfo.design.set_variable('Lj', '10 nH')
pinfo.design.set_variable('Cj', '2 fF')

print(f'''Full-chip eigenmode setup :
  n_modes = {setup.n_modes}
  passes  = {setup.passes}
  delta_f = {setup.delta_f}%
''')

setup.analyze()
print('Full-chip eigenmode simulation completed.')

# Convergence check -- 5 modes sous delta_f = 0.5%
eig_full.sim.convergence_t, eig_full.sim.convergence_f, _ = hfss.get_convergences()
eig_full.sim.plot_convergences()

# EPR -- DEUX jonctions pour l'analyse full-chip
```

```
pinfo = hfss.pinfo
pinfo.junctions.clear()   # on repart de zéro pour ce design

pinfo.junctions['jj_Q1'] = {
    'Lj_variable': 'Lj',
    'rect':        'JJ_rect_Lj_Q1_rect_jj',
    'line':        'JJ_Lj_Q1_rect_jj_',
    'Cj_variable': 'Cj',
}
pinfo.junctions['jj_Q2'] = {
    'Lj_variable': 'Lj',
    'rect':        'JJ_rect_Lj_Q2_rect_jj',
    'line':        'JJ_Lj_Q2_rect_jj_',
    'Cj_variable': 'Cj',
}
pinfo.validate_junction_info()

# Substrat = source principale de dissipation diélectrique
pinfo.dissipative['dielectrics_bulk'] = ['main']

eprd_full = epr.DistributedAnalysis(pinfo)

E_elec_full           = eprd_full.calc_energy_electric()
E_elec_substrate_full = eprd_full.calc_energy_electric(None, 'main')
E_mag_full            = eprd_full.calc_energy_magnetic()

print(f'''Full-chip -- energy distribution :
  E_elec_total     = {E_elec_full:.4e} J
  E_elec_substrate = {E_elec_substrate_full:.4e} J  ({100*E_elec_substrate_full/E_elec_full:.1f
    }%)
  E_mag_total      = {E_mag_full:.4e} J
''')

# Analyse EPR complète → participation matrix, cross-Kerr
try:
    eprd_full.do_EPR_analysis()
except Exception as _e:
    print(f'  [!] com_error cosmétique ignorée : {type(_e).__name__}')
    print('  → les résultats physiques sont OK, on continue.')
eprh_full = epr.QuantumAnalysis(eprd_full.data_filename)
eprh_full.analyze_all_variations(cos_trunc=10, fock_trunc=8)
eprh_full.plot_hamiltonian_results()

# Identification des modes physiques via les participations de jonction :
#   - le mode avec la plus grande participation de jj_Q1 = mode Q1
#   - idem pour jj_Q2
#   - les modes restants (participation ≈ 0 aux jonctions) = résonateurs / feedline
try:
    participations = eprh_full.get_participations()
    print('Participations (mode × junction) :')
    print(participations)
except Exception as e:
    print(f'get_participations indispo dans ta version pyEPR : {e}')
    print('Regarde manuellement eprh_full.results[last_var] pour identifier les modes.')

# Cleanup
```

```
hfss.clean_active_design()
hfss.stop()

from qiskit_metal.analyses.quantization import LOManalysis

# LOM analysis avec le renderer Q3D
c_lom = LOManalysis(design, "q3d")
q3d   = c_lom.sim.renderer

q3d.start()
q3d.activate_ansys_design("FullChip_Q3D", 'capacitive')

# Rendu complet du chip en Q3D -- modèle DIRECT.
# PLUS de pin ouvert sur les qubits : tous les pads sont connectés à un
# composant ou à la masse. Le couplage Q1↔Q2 est DIRECT (capa mutuelle des
# claws + CPW straight de 2 mm) -- il apparaîtra en gros terme hors-diagonale
# entre les nets pad_top_Q1 / pad_bot_Q1 et cross_Q2.
q3d.render_design(
    selection = [
        'Q1', 'Q2',
        # plus de Coupler_Q1Q2 -- le coupler est intégré dans Q2 via gap_extend_length
        'TQ1', 'TQ2', 'Bus1', 'Bus_to_Q2',
        'cpw_eastmost', 'cpw_middle', 'cpw_westmost',
        'launch_TL_L', 'launch_TL_R', 'launch_CL',
    ],
    open_pins = [],   # Q1.b est aussi connecté au coupler intégré dans Q2 -- plus de pin
    ouvert
)

# --- Q3D setup -- workaround pour bug com_error sur set_prop ---------------
# Bug pyEPR 0.8.5.7 / Ansys 2025 R1 : q3d_setup.passes = 20 déclenche une
# com_error via ChangeProperty. Solution : passer par l'API AEDT native.
try:
    oDesign = q3d.pinfo.design._design
    oAnalysisModule = oDesign.GetModule("AnalysisSetup")
    oAnalysisModule.EditSetup("Setup", [
        "NAME:Setup",
        "AdaptiveFreq:=", "1GHz",
        "SaveFields:=", False,
        "Enabled:=", True,
        ["NAME:Cap",
         "MaxPass:=",     20,
         "MinPass:=",     10,
         "MinConvPass:=", 2,
         "PerError:=",    0.5,
         "PerRefine:=",   30,
        ]
    ])
    print('✓ Q3D setup configuré via API native AEDT.')
except Exception as e:
    print(f'[!] EditSetup a raté : {type(e).__name__}: {e}')
    print('  → Configure manuellement dans AEDT :')
    print('    Project Manager → Analysis → Setup (double-clic)')
    print('    MaxPass=20, MinPass=10, MinConvPass=2, PerError=0.5%')

# Solve électrostatique Q3D
```

```
q3d.analyze_setup("Setup")
print('Q3D capacitance extraction completed.')

# Récupération de la matrice de capacité + convergence
c_lom.sim.capacitance_matrix, c_lom.sim.units = q3d.get_capacitance_matrix()

print(f'\nMatrice de capacité [{c_lom.sim.units}] :')
print(c_lom.sim.capacitance_matrix.round(3))
print(f'\nShape : {c_lom.sim.capacitance_matrix.shape}')

c_lom.sim.capacitance_all_passes, _ = q3d.get_capacitance_all_passes()
c_lom.sim.convergence_t, c_lom.sim.convergence_f, _ = q3d.get_convergences()
c_lom.sim.plot_convergences()

# -- Inspection des nets Q3D et identification des îlots --------------------
cap_matrix = c_lom.sim.capacitance_matrix   # pd.DataFrame en fF

print('Nets disponibles dans la matrice Q3D :')
for n in cap_matrix.index:
    print(f'  {n!r}')

# Nommage typique qiskit-metal :
#   TransmonPocketCL   → 'pad_top_Q1', 'pad_bot_Q1'     (JJ = différentiel)
#   TransmonCross      → 'cross_Q2'                      (JJ = single-ended vs GND)
q1_top_pad = 'pad_top_Q1'
q1_bot_pad = 'pad_bot_Q1'
# Détection auto du nom de l'îlot Q2 (modèle direct : cross_Q2 a fusionné
# avec c_connector_arm_Q2 via l'extension du pad coupleur)
q2_candidates = ['cross_Q2', 'c_connector_arm_Q2']
q2_island = next((n for n in q2_candidates if n in cap_matrix.index), None)
if q2_island is None:
    raise KeyError(f"Aucun net Q2 trouvé parmi {q2_candidates}. "
                   f"Disponibles : {list(cap_matrix.index)}")
print(f"Îlot Q2 identifié comme : {q2_island}")

for name in (q1_top_pad, q1_bot_pad, q2_island):
    if name not in cap_matrix.index:
        raise KeyError(f"Net {name!r} introuvable. "
                       f"Disponibles : {list(cap_matrix.index)}")

# -- CORRECTION 1 : élimination du net de masse -----------------------------
# Les résonateurs lambda/4 sont court-circuités à la masse à une extrémité.
# Q3D étant électrostatique, il ne peut pas distinguer le résonateur du plan
# de masse : il les fusionne en UN SEUL net, qu'il nomme d'après l'un des
# morceaux ('a_connector_pad_Q1' ici). Ce net EST donc le plan de masse.
# Il doit être éliminé (suppression de sa ligne et de sa colonne) pour
# devenir le datum du circuit ; sinon les charges de Q2 sont référencées à
# l'infini et non à la masse, et EC_2 est surestimé de ~40 %.
#
# Signature de détection : le net qui est le partenaire capacitif dominant
# de tous les autres, et après lequel la capacité résiduelle vers l'infini
# de chacun des autres nets est ~0.
def detect_ground_net(cap_df):
    nets = list(cap_df.index)
    V = cap_df.values
    votes = {n: 0 for n in nets}
```

```
    for i in range(len(nets)):
        mut = {nets[j]: -V[i, j] for j in range(len(nets)) if j != i}
        votes[max(mut, key=mut.get)] += 1
    cand, nv = max(votes.items(), key=lambda kv: kv[1])
    return cand if nv >= (len(nets) - 1) / 2 else None

gnd_net = detect_ground_net(cap_matrix)
if gnd_net is not None and gnd_net not in (q1_top_pad, q1_bot_pad, q2_island):
    print(f"Net de masse identifié : {gnd_net!r} → éliminé (datum du circuit)")
    keep = [n for n in cap_matrix.index if n != gnd_net]
    cap_matrix = cap_matrix.loc[keep, keep]
else:
    print("Aucun net de masse identifié -- datum laissé à l'infini")

print('\nLes 3 nets qubit ont été trouvés -- on peut construire le LOM.')

# -- LOM 2 qubits à la main depuis la matrice inverse de capacité ---------
# On contourne LOManalysis.run_lom() qui ne gère qu'une jonction.
#
# Physique :
#   1. Construire C en F, en ajoutant Cj sur chaque jonction :
#        - Q1 : Cj entre pad_top et pad_bot (mode différentiel)
#        - Q2 : Cj entre cross_island et gnd (mode single-ended)
#   2. Inverser → C^-1 (matrice inverse de capacité de Maxwell)
#   3. Vecteurs de mode :
#        v_Q1 = e_top - e_bot      (NON normalisé, cf. CORRECTION 2)
#        v_Q2 = e_cross
#   4. (C^-1)_ii = v_i^T C^-1 v_i          → capacité effective inverse par qubit
#      (C^-1)_12 = v_1^T C^-1 v_2
#   5. EC_i = e^2(C^-1)_ii / (2h)
#      EJ_i = (Φ_0/2π)^2 / (Lj_i · h)
#      f_lin_i = √(8 EJ_i EC_i)              (mode linéarisé = sortie HFSS)
#      ω_q_i   = √(8 EJ_i EC_i) - EC_i       (approx transmon)
#      α_i     = -EC_i
#   6. g_12 = 1/2 · (C^-1)_12 / √((C^-1)_11·(C^-1)_22) · √(ω_q1 · ω_q2)

import numpy as np
from scipy.constants import e, h, hbar, pi

Lj1 = 10e-9   # H
Lj2 = 10e-9   # H
Cj  = 2e-15   # F  (jonction shunt sur les deux qubits)
PHI_0 = h / (2*e)   # flux quantum
PHI_R = PHI_0 / (2*pi)

# 1) construire C en F (Q3D est en fF)
C = cap_matrix.copy().values.astype(float) * 1e-15  # fF → F
nets = list(cap_matrix.index)

def idx(name):
    return nets.index(name)

i_top, i_bot, i_cross = idx(q1_top_pad), idx(q1_bot_pad), idx(q2_island)

# Ajout des Cj -- jonctions en parallèle avec la capacité géométrique :
# Q1 : Cj entre top et bot → ajoute Cj à C[top,top] et C[bot,bot],
```

```
#      retire Cj sur les off-diag correspondantes
C[i_top, i_top]  += Cj
C[i_bot, i_bot]  += Cj
C[i_top, i_bot]  -= Cj
C[i_bot, i_top]  -= Cj
# Q2 : Cj entre cross et gnd (déjà implicite : masse = référence).
C[i_cross, i_cross] += Cj

# 2) inverse
Cinv = np.linalg.inv(C)

# 3) vecteurs de mode
# -- CORRECTION 2 : le vecteur différentiel ne doit PAS être normalisé ------
# Pour deux pads reliés par une capacité C_s, on a l'identité exacte
#     (e_top - e_bot)^T C^-1 (e_top - e_bot) = 1 / C_s ,
# c'est-à-dire la capacité différentielle physique du transmon flottant.
# Insérer un 1/√2 divise (C^-1)_11 par 2, donc EC_1 par 2 et f_lin par √2.
# (g_12 n'est pas affecté : la formule de l'étape 6 est invariante d'échelle.)
N = C.shape[0]
v_Q1 = np.zeros(N); v_Q1[i_top] = 1.0; v_Q1[i_bot] = -1.0
v_Q2 = np.zeros(N); v_Q2[i_cross] = 1.0

# 4) capa effective inverse
Cinv_11 = v_Q1 @ Cinv @ v_Q1
Cinv_22 = v_Q2 @ Cinv @ v_Q2
Cinv_12 = v_Q1 @ Cinv @ v_Q2

# 5) énergies + fréquences
EC1_J = (e**2) * Cinv_11 / 2
EC2_J = (e**2) * Cinv_22 / 2
EJ1_J = PHI_R**2 / Lj1
EJ2_J = PHI_R**2 / Lj2

# en Hz (÷h) puis GHz
EC1 = EC1_J / h / 1e9
EC2 = EC2_J / h / 1e9
EJ1 = EJ1_J / h / 1e9
EJ2 = EJ2_J / h / 1e9

f_lin1 = np.sqrt(8*EJ1*EC1)              # GHz -- mode linéarisé (= HFSS)
f_lin2 = np.sqrt(8*EJ2*EC2)              # GHz
omega_q1 = f_lin1 - EC1                  # GHz
omega_q2 = f_lin2 - EC2                  # GHz
alpha_1  = -EC1 * 1e3                    # MHz
alpha_2  = -EC2 * 1e3                    # MHz
C_sigma1 = e**2 / (2 * EC1_J) * 1e15     # fF
C_sigma2 = e**2 / (2 * EC2_J) * 1e15     # fF

# 6) couplage qubit-qubit
g_12 = 0.5 * Cinv_12 / np.sqrt(Cinv_11 * Cinv_22) \
       * np.sqrt(omega_q1 * omega_q2) * 1e3      # MHz

print('=== LOM 2-qubits (custom) ===')
print(f'  EC_1     = {EC1*1e3:.3f} MHz     EC_2     = {EC2*1e3:.3f} MHz')
print(f'  EJ_1     = {EJ1:.3f} GHz       EJ_2     = {EJ2:.3f} GHz')
print(f'  EJ/EC 1  = {EJ1/EC1:.3f}        EJ/EC 2  = {EJ2/EC2:.3f}')
```

```
print(f'  f_lin1   = {f_lin1:.3f} GHz     f_lin2   = {f_lin2:.3f} GHz')
print(f'  ω_q1     = {omega_q1:.3f} GHz     ω_q2     = {omega_q2:.3f} GHz')
print(f'  α_1      = {alpha_1:.3f} MHz     α_2      = {alpha_2:.3f} MHz')
print(f'  CΣ_1     = {C_sigma1:.3f} fF      CΣ_2     = {C_sigma2:.3f} fF')
print(f'  g_12    ≈ {g_12:.3f} MHz')

# Cleanup Q3D
q3d.clean_active_design()
q3d.stop()
```

Quantum processor implementation — Indirect coupling model

```
#!/usr/bin/env python
# coding: utf-8
"""
Quantum processor implementation -- INDIRECT coupling model
Master thesis annex : qiskit-metal design + HFSS/Q3D/EPR simulation pipeline.
Two transmons (TransmonPocketCL Q1 + TransmonCross Q2), each with its own
CoupledLineTee readout on a shared feedline. Q1<->Q2 coupling is mediated
by the feedline mode (no direct capacitor between qubits).
"""


# ============================================================================
# Annexe B (Indirect Coupling Model) -- Quantum Processor with Two-Tee Readout
# ============================================================================

# ============================================================================
# Environment Initialization
# ============================================================================

# Enable automatic reloading of modules when modifications are made
get_ipython().run_line_magic('load_ext', 'autoreload')
get_ipython().run_line_magic('autoreload', '2')

import qiskit_metal as metal
from qiskit_metal import designs, draw
from qiskit_metal import Dict, open_docs, Headings

# Electromagnetic analysis toolkit
import pyEPR as epr

# Numerical / plotting
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Ansys renderer (for default options inspection)
from qiskit_metal.renderers.renderer_ansys.ansys_renderer import QAnsysRenderer
print('pyEPR :', repr(epr.__version__))


# DesignPlanar : 2D circuit architecture for superconducting qubits
design = designs.DesignPlanar()
```

```
43
44 # Chip sizing -- extended along x to host Q1 + coupler + Q2 + readout subsystem
45 # Default DesignPlanar chip is ~9 x 6 mm. We explicitly set it for reproducibility.
46 design.chips.main.size.size_x = '8 mm'
47 design.chips.main.size.size_y = '6 mm'
48
49 # Enable design overwrite (iterative refinement without object collisions)
50 design.overwrite_enabled = True
51
52 # ============================================================================
53 # Quantum Component Library Import
54 # ============================================================================
55
56 # Qubits
57 # TransmonPocketCL : pocket-style transmon with built-in charge line (CL) stub
58 from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL
59
60 # TransmonCross : cross-shaped (Xmon) transmon with up to 3 connection pads
61 from qiskit_metal.qlibrary.qubits.transmon_cross import TransmonCross
62
63 # Waveguide routing
64 from qiskit_metal.qlibrary.tlines.straight_path import RouteStraight    # Direct CPW
65 from qiskit_metal.qlibrary.tlines.meandered import RouteMeander          # Meandered (resonator
       / bus)
66 from qiskit_metal.qlibrary.tlines.pathfinder import RoutePathfinder      # Auto-router fallback
67 from qiskit_metal.qlibrary.tlines.anchored_path import RouteAnchors      # Manual waypoints
68
69 # Couplers / tees
70 from qiskit_metal.qlibrary.couplers.coupled_line_tee import CoupledLineTee  # Capacitive tee (
       readout)
71 from qiskit_metal.qlibrary.couplers.line_tee         import LineTee         # Galvanic tee (Y-
       node)
72
73 # Terminations
74 from qiskit_metal.qlibrary.terminations.open_to_ground import OpenToGround
75 from qiskit_metal.qlibrary.terminations.short_to_ground import ShortToGround
76
77 # Launch pads (wirebond interface to chip carrier)
78 from qiskit_metal.qlibrary.terminations.launchpad_wb import LaunchpadWirebond
79
80 # ============================================================================
81 # Custom Component -- `TransmonPocketCL_LongStub`
82 # ============================================================================
83
84 # Subclass de TransmonPocketCL avec cl_cpw_length tunable
85 from qiskit_metal.qlibrary.qubits.transmon_pocket_cl import TransmonPocketCL
86 from qiskit_metal import draw, Dict
87
88
89 class TransmonPocketCL_LongStub(TransmonPocketCL):
90     """TransmonPocketCL avec la longueur du cl_cpw exposée comme paramètre.
91
92     Le composant parent hard-code la longueur de cl_cpw à `8 * cl_width`.
93     Cette sous-classe expose un nouveau paramètre `cl_cpw_length` (par défaut
94     '200 um' = ~2.5x la valeur d'origine de 80 µm). Permet d'allonger le stub
95     interne sans toucher à `cl_width` et donc sans changer l'impédance.
```

```
    """

    default_options = Dict(
        **TransmonPocketCL.default_options,
        cl_cpw_length='200um',
    )

    def make_charge_line(self):
        """Override : utilise cl_cpw_length au lieu de 8*cl_width."""
        name = 'Charge_Line'
        p = self.p
        L = p.cl_cpw_length

        cl_arm = draw.box(0, 0, -p.cl_width, p.cl_length)
        cl_cpw = draw.box(0, 0, -L, p.cl_width)
        cl_metal = draw.unary_union([cl_arm, cl_cpw])
        cl_etcher = draw.buffer(cl_metal, p.cl_gap)

        port_line = draw.LineString([(-L, 0), (-L, p.cl_width)])

        polys = [cl_metal, cl_etcher, port_line]

        cl_rotate = 0
        if (abs(p.cl_pocket_edge) > 135) or (abs(p.cl_pocket_edge) < 45):
            polys = draw.translate(
                polys, -(p.pocket_width / 2 + p.cl_ground_gap + p.cl_gap),
                p.cl_off_center)
            if (abs(p.cl_pocket_edge) > 135):
                cl_rotate = 180
        else:
            polys = draw.translate(
                polys, -(p.pocket_height / 2 + p.cl_ground_gap + p.cl_gap),
                p.cl_off_center)
            cl_rotate = 90
            if (p.cl_pocket_edge < 0):
                cl_rotate = -90

        polys = draw.rotate(polys, p.orientation + cl_rotate, origin=(0, 0))
        polys = draw.translate(polys, p.pos_x, p.pos_y)
        [cl_metal, cl_etcher, port_line] = polys

        points = list(draw.shapely.geometry.shape(port_line).coords)
        self.add_pin(name, points, p.cl_width)
        self.add_qgeometry('poly', dict(cl_metal=cl_metal))
        self.add_qgeometry('poly', dict(cl_etcher=cl_etcher), subtract=True)

# ============================================================================
# A.1 -- Q1 : TransmonPocketCL with bias line *(charge line + readout pad + coupler pad)*
# ============================================================================

# Q1 -- pocket transmon with integrated charge line.
# Two connection pads on the pocket :
#   - 'a' : facing the readout resonator (north of Q1 after the 90° orientation)
#   - 'b' : facing Q2 (the Xmon, placed east of Q1) -- used for the direct coupler
#
# Orientation note : Q1 is rotated 90° in the world frame.
```

```
# - 'a' at connector_location='180' → routes toward TQ1 (north of Q1).
# - 'b' at connector_location='0'   → opposite side of the pocket, faces Q2.
# - cl_pocket_edge='0'              → charge line stub exits south (toward launch_CL).
import os
design.overwrite_enabled = True

Q1 = TransmonPocketCL_LongStub(design, 'Q1', options=dict(
    pos_x='0 mm',
    pos_y='0 mm',
    orientation='90',

    # Two connection pads -- pocket sides
    connection_pads=dict(
        a=dict(connector_location='180', claw_length='95 um'),  # → readout (TQ1)
        b=dict(connector_location='0',   claw_length='95 um'),  # → direct coupler (Q2)
    ),

    # CL stub geometry -- unchanged from the original design
    make_CL=True,            # Enable the integrated charge line stub
    cl_pocket_edge='0',      # CL stub side : east in local frame ⇒ south in world frame
    cl_off_center='50 um',   # Offset from pocket symmetry axis
    cl_width='10 um',        # CL CPW center-conductor width
    cl_gap='6 um',           # CL CPW gap to ground plane
    cl_length='20 um',       # CL stub protrusion into pocket
    cl_ground_gap='6 um',    # Gap at the open end of the stub
    cl_cpw_length='1.138 mm',  # stub long (=1.1 mm original + 38 µm) : la fin du stub tombe
    pile sur launch_CL.tie (y=-1.475 mm). Aucune CPW externe cpw_CL nécessaire ; l'etcher du
    pocket ne recouvre pas la CPW.
))

design.rebuild()

# ============================================================================
# A.1ter -- Custom Component -- `TransmonCross_LongC`
# ============================================================================

# Sous-classe de TransmonCross avec gap_extend_length exposé
from qiskit_metal.qlibrary.qubits.transmon_cross import TransmonCross
from qiskit_metal import draw, Dict


class TransmonCross_LongC(TransmonCross):
    """TransmonCross avec la longueur du bras type='1' (gap) exposée comme paramètre.

    Le composant parent hard-code la longueur du connector_arm de type '1' à
    `4 * claw_width` (= 40 µm avec claw_width=10 µm). Cette sous-classe expose
    un paramètre par-pad `gap_extend_length`. Quand il vaut None (défaut), le
    comportement d'origine est conservé. Sinon, le bras métal du pad type-'1'
    est allongé à cette valeur -- permet de rapprocher la pointe du pad d'un
    autre composant pour former un direct coupler capacitif sans instancier
    une CPW externe.

    Convention littérature : ~20 µm de gap capacitif nu entre la pointe de
    l'extension et le métal voisin (typiquement la pastille d'un pocket
    transmon).
    """
```

```
206
207     default_options = Dict(**TransmonCross.default_options)
208     default_options['_default_connection_pads'] = Dict(
209     **TransmonCross.default_options['_default_connection_pads'],
210     gap_extend_length   = None,
211     claw_width_back     = None,   # ← évite le warning
212     ground_spacing_back = None,   # ← évite le warning
213 )
214
215     default_options = Dict(**TransmonCross.default_options)
216     default_options['_default_connection_pads'] = Dict(
217         **TransmonCross.default_options['_default_connection_pads'],
218         gap_extend_length=None,
219         claw_width_back=None,
220         ground_spacing_back=None,
221     )
222
223     def make_connection_pad(self, name: str):
224         """Override : bras type='1' allongeable via gap_extend_length."""
225         p = self.p
226         cross_width  = p.cross_width
227         cross_length = p.cross_length
228         cross_gap    = p.cross_gap
229         chip = p.chip
230
231         pc    = p.connection_pads[name]
232         c_g   = pc.claw_gap
233         c_l   = pc.claw_length
234         c_w   = pc.claw_width
235         c_c_w = pc.claw_cpw_width
236         c_c_l = pc.claw_cpw_length
237         g_s   = pc.ground_spacing
238         con_loc = pc.connector_location
239
240         c_w_b = pc.claw_width_back if pc.claw_width_back is not None else c_w
241         g_s_b = pc.ground_spacing_back if pc.ground_spacing_back is not None else g_s
242
243         claw_cpw = draw.box(-c_w_b, -c_c_w/2, -c_c_l - c_w_b, c_c_w/2)
244
245         if pc.connector_type == 0:
246             t_claw_height = 2*c_g + 2*c_w + 2*g_s + 2*cross_gap + cross_width
247             claw_base = draw.box(-c_w_b, -t_claw_height/2, c_l, t_claw_height/2)
248             claw_subtract = draw.box(0, -t_claw_height/2 + c_w, c_l, t_claw_height/2 - c_w)
249             claw_base = claw_base.difference(claw_subtract)
250             connector_arm    = draw.shapely.ops.unary_union([claw_base, claw_cpw])
251             connector_etcher = draw.buffer(connector_arm, c_g)
252         else:
253             arm_len = pc.gap_extend_length if pc.gap_extend_length is not None else 4 * c_w
254             connector_arm    = draw.box(0, -c_w/2, -arm_len, c_w/2)
255             connector_etcher = draw.buffer(connector_arm, c_g)
256
257         port_line = draw.LineString([
258             (-c_c_l - c_w_b, -c_c_w/2),
259             (-c_c_l - c_w_b,  c_c_w/2),
260         ])
261
```

```
        # ← NOUVELLE logique (identique au fichier de référence GUI)
        claw_rotate = 0
        if con_loc > 135:
            claw_rotate = 180   # connector_location='180' → bras EST ✓
        elif con_loc > 45:
            claw_rotate = -90   # connector_location='90'  → bras NORD ✓

        polys = [connector_arm, connector_etcher, port_line]
        polys = draw.translate(polys, -(cross_length + cross_gap + g_s_b + c_g), 0)
        polys = draw.rotate(polys, claw_rotate, origin=(0, 0))
        polys = draw.rotate(polys, p.orientation, origin=(0, 0))
        polys = draw.translate(polys, p.pos_x, p.pos_y)
        [connector_arm, connector_etcher, port_line] = polys

        self.add_qgeometry("poly", {f"{name}_connector_arm": connector_arm}, chip=chip)
        self.add_qgeometry("poly", {f"{name}_connector_etcher": connector_etcher},
                           subtract=True, chip=chip)
        self.add_pin(name, port_line.coords, c_c_w)

# ============================================================================
# A.2 -- Q2 : Xmon (TransmonCross), raised to y = 0.8 mm
# ============================================================================

import os
design.overwrite_enabled = True

Q2 = TransmonCross_LongC(design, 'Q2', options=dict(
    pos_x='-2 mm', pos_y='0 mm', orientation='0',
    cross_width='20 um', cross_length='200 um', cross_gap='20 um',
    connection_pads=dict(
        c=dict(
            connector_location='180', connector_type='1',
            claw_length='30 um', claw_width='10 um', claw_gap='6 um',
            ground_spacing='54 um',
            gap_extend_length='50 um',   # ← contact métallique avec cpw_CL_Q2 (défaut hardcod
    é = 4×claw_width = 40 µm)
        ),  # east arm -- CL gap type ; g_s=54 µm ⇒ gap total = cross_gap(20)+g_s(54)+claw_gap
    (6) = 80 µm
        #a=dict(connector_location='0',   connector_type='0', claw_length='30 um', claw_width
    ='10 um', claw_gap='6 um'),
        b=dict(connector_location='90',  connector_type='0', claw_length='30 um', claw_width='
    10 um', claw_gap='6 um'),
    ),
))

design.rebuild()

# ============================================================================
# A.4 -- Two readout subsystems  *(TQ1+Bus1 for Q1, TQ2+Bus_to_Q2 for Q2)*
# ============================================================================

import numpy as np

# -- Monkey-patch the qiskit-metal 0.1.5 collision-check bug -----------------
# parse_value() does NOT convert 'true'/'false' strings to Python booleans, so
# the option advanced.avoid_collision='false' (the default!) is truthy and the
```

```
# collision check runs anyway, hitting the AttributeError on MultiPolygon
# components. We force-bypass it once per kernel session.
from qiskit_metal.qlibrary.tlines import anchored_path as _ap
def _always_unobstructed(self, segment):
    return True
_ap.RouteAnchors.unobstructed = _always_unobstructed
try:
    from qiskit_metal.qlibrary.tlines import meandered as _md
    _md.RouteMeander.unobstructed = _always_unobstructed
except (ImportError, AttributeError):
    pass
try:
    from qiskit_metal.qlibrary.tlines import pathfinder as _pf
    _pf.RoutePathfinder.unobstructed = _always_unobstructed
except (ImportError, AttributeError):
    pass
# ----------------------------------------------------------------------------

design.overwrite_enabled = True

# 1) TQ1 -- readout tee for Q1 (unchanged geometry)
TQ1 = CoupledLineTee(design, 'TQ1', options=dict(
    pos_x='-0.5 mm',
    pos_y='2 mm',
    coupling_length='200 um',
    open_termination=False,
))

# 2) Bus1 -- readout meander Q1.a → TQ1.second_end
bus1 = RouteMeander(design, 'Bus1', options=dict(
    total_length    = '4.1 mm',
    hfss_wire_bonds = True,
    fillet          = '99 um',
    advanced=Dict(avoid_collision='false'),
    pin_inputs=Dict(
        start_pin=Dict(component='Q1',  pin='a'),
        end_pin  =Dict(component='TQ1', pin='second_end'),
    ),
))

# 3) TQ2 -- the NEW readout tee for Q2 (same geometry as TQ1)
TQ2 = CoupledLineTee(design, 'TQ2', options=dict(
    pos_x='-2 mm',
    pos_y='2 mm',
    coupling_length='200 um',
    open_termination=False,
))

Bus_to_Q2 = RouteMeander(design, 'Bus_to_Q2', options=dict(
    total_length    = '4.1 mm',
    hfss_wire_bonds = True,
    fillet          = '50 um',
    advanced        = Dict(avoid_collision='false'),
    lead            = Dict(
        start_straight = '150 um',   # ← segment droit forcé avant le 1er virage
        end_straight   = '100 um',
```

```
    ),
    pin_inputs=Dict(
        start_pin=Dict(component='Q2',  pin='b'),
        end_pin  =Dict(component='TQ2', pin='second_end'),
    ),
))

design.rebuild()

# ============================================================================
# A.5 -- Control & readout interface  *(feedline through both tees, no separate drive line)*
# ============================================================================

# Launch pads -- interface to the chip carrier (wirebond pads).
design.overwrite_enabled = True

launch_TL_L = LaunchpadWirebond(design, 'launch_TL_L', options=dict(
    pos_x='3 mm', pos_y='2 mm', orientation='180',
))
launch_TL_R = LaunchpadWirebond(design, 'launch_TL_R', options=dict(
    pos_x='-3 mm', pos_y='2 mm',
))

launch_CL = LaunchpadWirebond(design, 'launch_CL', options=dict(
    pos_x='-0.055 mm',
    pos_y='-1.5 mm',
    orientation='90',
))

# Feedline east : launch_TL_L → TQ1.prime_end
cpw_eastmost = RouteStraight(design, 'cpw_eastmost', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='launch_TL_L', pin='tie'),
        end_pin  =Dict(component='TQ1',         pin='prime_end'),
    ),
))

# Feedline middle : TQ1.prime_start → TQ2.prime_end
cpw_middle = RouteStraight(design, 'cpw_middle', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='TQ1', pin='prime_start'),
        end_pin  =Dict(component='TQ2', pin='prime_end'),
    ),
))

# Feedline west : TQ2.prime_start → launch_TL_R
cpw_westmost = RouteStraight(design, 'cpw_westmost', options=Dict(
    hfss_wire_bonds=True,
    pin_inputs=Dict(
        start_pin=Dict(component='TQ2',         pin='prime_start'),
        end_pin  =Dict(component='launch_TL_R', pin='tie'),
    ),
))

```

```
# Drive line Q1 -- pas de CPW externe.
# Le stub interne du Q1 (cl_cpw_length='1.138 mm') se termine exactement sur
# launch_CL.tie (y=-1.475 mm), donc les métaux se touchent et fusionnent
# naturellement dans le rendu GDS/HFSS.

# ============================================================================
# CHARGE LINE pour Q2 -- couplage capacitif au bras EST de l'Xmon (pad 'c')
# ============================================================================
# CPW en L : Q2.c (pin à x=-1.64, oriente EST) → sort à l'est → coin en (-1.4, 0) → descend sud
# Segment horizontal réduit à ~240 µm (au lieu de ~640 µm d'origine).

design.overwrite_enabled = True

# Launchpad drive line Q2 -- juste sous Q2, aligné en x avec l'ancre
launch_CL_Q2 = LaunchpadWirebond(design, 'launch_CL_Q2', options=dict(
    pos_x       = '-1.4 mm',    # à l'EST du pin Q2.c (qui est à x=-1.64) -- le pin pointe vers l'est
    pos_y       = '-1.5 mm',    # aligné en y avec launch_CL de Q1
    orientation = '90',         # tie points north -- l'L vertical y arrive par le nord
))

from qiskit_metal.qlibrary.tlines.anchored_path import RouteAnchors
from collections import OrderedDict
import numpy as np

cpw_CL_Q2 = RouteAnchors(design, 'cpw_CL_Q2', options=Dict(
    hfss_wire_bonds = False,
    fillet          = '20 um',
    lead            = Dict(start_straight='30 um', end_straight='30 um'),
    anchors         = OrderedDict({
        0: np.array([-1.4, 0.0]),    # coin à l'est de Q2.c → sortie propre du pin, puis descente sud
    }),
    advanced   = Dict(avoid_collision='false'),
    pin_inputs = Dict(
        start_pin=Dict(component='Q2',           pin='c'),
        end_pin  =Dict(component='launch_CL_Q2', pin='tie'),
    ),
))

design.rebuild()

# ============================================================================
# A.6 -- Final layout overview
# ============================================================================

# Full processor -- all components.
design.rebuild()
print('Components in design :')
for name, comp in design.components.items():
    print(f'  • {name:<20s} ({type(comp).__name__})')

# ============================================================================
# B -- Full-chip eigenmode simulation & EPR analysis
# ============================================================================
```

```
479
480 from qiskit_metal.analyses.quantization import EPRanalysis
481
482 # EPR analysis with the HFSS eigenmode solver on the full-chip design
483 eig_full = EPRanalysis(design, "hfss")
484 hfss     = eig_full.sim.renderer
485
486 # Launch Ansys Electronics Desktop and open a fresh eigenmode design container
487 hfss.start()
488 hfss.activate_ansys_design("FullChip_Eigenmode", 'eigenmode')
489
490 # Full chip render (Q1, Q2, both tees, both readout meanders, feedline segments,
491 # launchpads, Q1 CL stub, Q2 CL). No open_pins needed : all qubit pads are
492 # either connected to a resonator or facing the feedline mode.
493 hfss.render_design()
494
495 # Multi-mode setup
496 setup = hfss.pinfo.setup
497 setup.nmodes     = 5    # Q1 + Q2 + TQ1_readout + TQ2_readout + feedline
498 setup.n_modes    = 5
499 setup.passes     = 18
500 setup.min_passes = 8
501 setup.delta_f    = 0.5
502
503 # Josephson junctions -- Lj / Cj partagés (renomme en Lj1 / Lj2 pour un sweep)
504 pinfo = hfss.pinfo
505 pinfo.design.set_variable('Lj', '10 nH')
506 pinfo.design.set_variable('Cj', '2 fF')
507
508 print(f'''Full-chip eigenmode setup :
509   n_modes = {setup.n_modes}
510   passes  = {setup.passes}
511   delta_f = {setup.delta_f}%
512 ''')
513
514 setup.analyze()
515 print('Full-chip eigenmode simulation completed.')
516
517 # Convergence
518 eig_full = EPRanalysis(design, "hfss")
519 eig_full.sim.renderer = hfss
520 eig_full.sim.convergence_t, eig_full.sim.convergence_f, _ = hfss.get_convergences()
521 eig_full.sim.plot_convergences()
522
523 # EPR -- DEUX jonctions pour l'analyse full-chip
524 pinfo = hfss.pinfo
525 pinfo.junctions.clear()   # on repart de zéro pour ce design
526
527 pinfo.junctions['jj_Q1'] = {
528     'Lj_variable': 'Lj',
529     'rect':        'JJ_rect_Lj_Q1_rect_jj',
530     'line':        'JJ_Lj_Q1_rect_jj_',
531     'Cj_variable': 'Cj',
532 }
533 pinfo.junctions['jj_Q2'] = {
534     'Lj_variable': 'Lj',
```

```
    'rect':        'JJ_rect_Lj_Q2_rect_jj',
    'line':        'JJ_Lj_Q2_rect_jj_',
    'Cj_variable': 'Cj',
}
pinfo.validate_junction_info()

# Substrat = source principale de dissipation diélectrique
pinfo.dissipative['dielectrics_bulk'] = ['main']

eprd_full = epr.DistributedAnalysis(pinfo)

E_elec_full           = eprd_full.calc_energy_electric()
E_elec_substrate_full = eprd_full.calc_energy_electric(None, 'main')
E_mag_full            = eprd_full.calc_energy_magnetic()

print(f'''Full-chip -- energy distribution :
  E_elec_total     = {E_elec_full:.4e} J
  E_elec_substrate = {E_elec_substrate_full:.4e} J  ({100*E_elec_substrate_full/E_elec_full:.1f}%)
  E_mag_total      = {E_mag_full:.4e} J
''')

# Identification des modes physiques via les participations de jonction :
#   - le mode avec la plus grande participation de jj_Q1 = mode Q1
#   - idem pour jj_Q2
#   - les modes restants (participation ≈ 0 aux jonctions) = résonateurs / feedline
try:
    participations = eprh_full.get_participations()
    print('Participations (mode × junction) :')
    print(participations)
except Exception as e:
    print(f'get_participations indispo dans ta version pyEPR : {e}')
    print('Regarde manuellement eprh_full.results[last_var] pour identifier les modes.')

# Cleanup
hfss.clean_active_design()
hfss.stop()

from qiskit_metal.analyses.quantization import LOManalysis

# LOM analysis avec le renderer Q3D
c_lom = LOManalysis(design, "q3d")
q3d   = c_lom.sim.renderer

q3d.start()
q3d.activate_ansys_design("FullChip_Q3D", 'capacitive')

# 1) Nettoyer le design Q3D existant
q3d.clean_active_design()

# 2) Re-render
q3d.render_design(
    selection = [
        'Q1', 'Q2',
        'TQ1', 'TQ2', 'Bus1', 'Bus_to_Q2',
        'cpw_eastmost', 'cpw_middle', 'cpw_westmost',
```

```
        'cpw_CL_Q2',
        'launch_TL_L', 'launch_TL_R', 'launch_CL', 'launch_CL_Q2',
    ],
    open_pins = [('Q1', 'b')],
)

q3d.add_q3d_setup(
    name          = "Setup",
    max_passes    = 20,
    min_passes    = 10,
    percent_error = 0.5,
    min_converged = 2,
)
print('Q3D setup configuré.')

# Solve électrostatique Q3D
q3d.analyze_setup("Setup")
print('Q3D capacitance extraction completed.')

# Récupération de la matrice de capacité + convergence
c_lom.sim.capacitance_matrix, c_lom.sim.units = q3d.get_capacitance_matrix()

print(f'\nMatrice de capacité [{c_lom.sim.units}] :')
print(c_lom.sim.capacitance_matrix.round(3))
print(f'\nShape : {c_lom.sim.capacitance_matrix.shape}')

c_lom.sim.capacitance_all_passes, _ = q3d.get_capacitance_all_passes()
c_lom.sim.convergence_t, c_lom.sim.convergence_f, _ = q3d.get_convergences()
c_lom.sim.plot_convergences()

# -- Inspection des nets Q3D et identification des îlots --------------------
cap_matrix = c_lom.sim.capacitance_matrix   # pd.DataFrame en fF

print('Nets disponibles dans la matrice Q3D :')
for n in cap_matrix.index:
    print(f'  {n!r}')

# Nommage typique qiskit-metal :
#   TransmonPocketCL   → 'pad_top_Q1', 'pad_bot_Q1'     (JJ = différentiel)
#   TransmonCross      → 'cross_Q2'                     (JJ = single-ended vs GND)
q1_top_pad = 'pad_top_Q1'
q1_bot_pad = 'pad_bot_Q1'
# Détection auto du nom de l'îlot Q2 (modèle direct : cross_Q2 a fusionné
# avec c_connector_arm_Q2 via l'extension du pad coupleur)
q2_candidates = ['cross_Q2', 'c_connector_arm_Q2']
q2_island = next((n for n in q2_candidates if n in cap_matrix.index), None)
if q2_island is None:
    raise KeyError(f"Aucun net Q2 trouvé parmi {q2_candidates}. "
                   f"Disponibles : {list(cap_matrix.index)}")
print(f"Îlot Q2 identifié comme : {q2_island}")

for name in (q1_top_pad, q1_bot_pad, q2_island):
    if name not in cap_matrix.index:
        raise KeyError(f"Net {name!r} introuvable. "
                       f"Disponibles : {list(cap_matrix.index)}")

```

```
# -- CORRECTION 1 : élimination du net de masse ----------------------------
# Les résonateurs lambda/4 sont court-circuités à la masse à une extrémité.
# Q3D étant électrostatique, il ne peut pas distinguer le résonateur du plan
# de masse : il les fusionne en UN SEUL net, qu'il nomme d'après l'un des
# morceaux ('a_connector_pad_Q1' ici). Ce net EST donc le plan de masse.
# Il doit être éliminé (suppression de sa ligne et de sa colonne) pour
# devenir le datum du circuit ; sinon les charges de Q2 sont référencées à
# l'infini et non à la masse, et EC_2 est surestimé de ~40 %.
#
# Signature de détection : le net qui est le partenaire capacitif dominant
# de tous les autres, et après lequel la capacité résiduelle vers l'infini
# de chacun des autres nets est ~0.
def detect_ground_net(cap_df):
    nets = list(cap_df.index)
    V = cap_df.values
    votes = {n: 0 for n in nets}
    for i in range(len(nets)):
        mut = {nets[j]: -V[i, j] for j in range(len(nets)) if j != i}
        votes[max(mut, key=mut.get)] += 1
    cand, nv = max(votes.items(), key=lambda kv: kv[1])
    return cand if nv >= (len(nets) - 1) / 2 else None

gnd_net = detect_ground_net(cap_matrix)
if gnd_net is not None and gnd_net not in (q1_top_pad, q1_bot_pad, q2_island):
    print(f"Net de masse identifié : {gnd_net!r} → éliminé (datum du circuit)")
    keep = [n for n in cap_matrix.index if n != gnd_net]
    cap_matrix = cap_matrix.loc[keep, keep]
else:
    print("Aucun net de masse identifié -- datum laissé à l'infini")

print('\nLes 3 nets qubit ont été trouvés -- on peut construire le LOM.')

# -- LOM 2 qubits à la main depuis la matrice inverse de capacité ---------
# On contourne LOManalysis.run_lom() qui ne gère qu'une jonction.
#
# Physique :
#   1. Construire C en F, en ajoutant Cj sur chaque jonction :
#        - Q1 : Cj entre pad_top et pad_bot (mode différentiel)
#        - Q2 : Cj entre cross_island et gnd (mode single-ended)
#   2. Inverser → C^-1 (matrice inverse de capacité de Maxwell)
#   3. Vecteurs de mode :
#        v_Q1 = e_top - e_bot      (NON normalisé, cf. CORRECTION 2)
#        v_Q2 = e_cross
#   4. (C^-1)_ii = v_i^T C^-1 v_i         → capacité effective inverse par qubit
#      (C^-1)_12 = v_1^T C^-1 v_2
#   5. EC_i = e^2(C^-1)_ii / (2h)
#      EJ_i = (Φ_0/2π)^2 / (Lj_i · h)
#      f_lin_i = √(8 EJ_i EC_i)              (mode linéarisé = sortie HFSS)
#      ω_q_i   = √(8 EJ_i EC_i) - EC_i       (approx transmon)
#      α_i     = -EC_i
#   6. g_12 = 1/2 · (C^-1)_12 / √((C^-1)_11·(C^-1)_22) · √(ω_q1 · ω_q2)

import numpy as np
from scipy.constants import e, h, hbar, pi

Lj1 = 10e-9   # H
```

```
Lj2 = 10e-9   # H
Cj  = 2e-15   # F  (jonction shunt sur les deux qubits)
PHI_0 = h / (2*e)   # flux quantum
PHI_R = PHI_0 / (2*pi)

# 1) construire C en F (Q3D est en fF)
C = cap_matrix.copy().values.astype(float) * 1e-15  # fF → F
nets = list(cap_matrix.index)

def idx(name):
    return nets.index(name)

i_top, i_bot, i_cross = idx(q1_top_pad), idx(q1_bot_pad), idx(q2_island)

# Ajout des Cj -- jonctions en parallèle avec la capacité géométrique :
# Q1 : Cj entre top et bot → ajoute Cj à C[top,top] et C[bot,bot],
#      retire Cj sur les off-diag correspondantes
C[i_top, i_top]  += Cj
C[i_bot, i_bot]  += Cj
C[i_top, i_bot]  -= Cj
C[i_bot, i_top]  -= Cj
# Q2 : Cj entre cross et gnd (déjà implicite : masse = référence).
C[i_cross, i_cross] += Cj

# 2) inverse
Cinv = np.linalg.inv(C)

# 3) vecteurs de mode
# -- CORRECTION 2 : le vecteur différentiel ne doit PAS être normalisé ------
# Pour deux pads reliés par une capacité C_s, on a l'identité exacte
#     (e_top - e_bot)^T C^{-1} (e_top - e_bot) = 1 / C_s ,
# c'est-à-dire la capacité différentielle physique du transmon flottant.
# Insérer un 1/√2 divise (C^{-1})_11 par 2, donc EC_1 par 2 et f_lin par √2.
# (g_12 n'est pas affecté : la formule de l'étape 6 est invariante d'échelle.)
N = C.shape[0]
v_Q1 = np.zeros(N); v_Q1[i_top] = 1.0; v_Q1[i_bot] = -1.0
v_Q2 = np.zeros(N); v_Q2[i_cross] = 1.0

# 4) capa effective inverse
Cinv_11 = v_Q1 @ Cinv @ v_Q1
Cinv_22 = v_Q2 @ Cinv @ v_Q2
Cinv_12 = v_Q1 @ Cinv @ v_Q2

# 5) énergies + fréquences
EC1_J = (e**2) * Cinv_11 / 2
EC2_J = (e**2) * Cinv_22 / 2
EJ1_J = PHI_R**2 / Lj1
EJ2_J = PHI_R**2 / Lj2

# en Hz (÷h) puis GHz
EC1 = EC1_J / h / 1e9
EC2 = EC2_J / h / 1e9
EJ1 = EJ1_J / h / 1e9
EJ2 = EJ2_J / h / 1e9

f_lin1 = np.sqrt(8*EJ1*EC1)              # GHz -- mode linéarisé (= HFSS)
```

```
f_lin2 = np.sqrt(8*EJ2*EC2)              # GHz
omega_q1 = f_lin1 - EC1                  # GHz
omega_q2 = f_lin2 - EC2                  # GHz
alpha_1  = -EC1 * 1e3                    # MHz
alpha_2  = -EC2 * 1e3                    # MHz
C_sigma1 = e**2 / (2 * EC1_J) * 1e15     # fF
C_sigma2 = e**2 / (2 * EC2_J) * 1e15     # fF

# 6) couplage qubit-qubit
g_12 = 0.5 * Cinv_12 / np.sqrt(Cinv_11 * Cinv_22) \
       * np.sqrt(omega_q1 * omega_q2) * 1e3      # MHz

print('=== LOM 2-qubits (custom) ===')
print(f'  EC_1     = {EC1*1e3:.3f} MHz     EC_2     = {EC2*1e3:.3f} MHz')
print(f'  EJ_1     = {EJ1:.3f} GHz       EJ_2     = {EJ2:.3f} GHz')
print(f'  EJ/EC 1  = {EJ1/EC1:.3f}        EJ/EC 2  = {EJ2/EC2:.3f}')
print(f'  f_lin1   = {f_lin1:.3f} GHz     f_lin2   = {f_lin2:.3f} GHz')
print(f'  ω_q1     = {omega_q1:.3f} GHz     ω_q2     = {omega_q2:.3f} GHz')
print(f'  α_1      = {alpha_1:.3f} MHz     α_2      = {alpha_2:.3f} MHz')
print(f'  CΣ_1     = {C_sigma1:.3f} fF      CΣ_2     = {C_sigma2:.3f} fF')
print(f'  g_12    ≈ {g_12:.3f} MHz')

# Cleanup Q3D
q3d.clean_active_design()
q3d.stop()
```

5.1 ZZ coupling

The ZZ coupling can appear as a parasitic effect that limits two-qubit gate fidelity in the direct-coupling geometry of Sec. 1.1.7, or as a useful entangling interaction between logical qubits in the DFS encoding of Fig. 2. The Hamiltonian interaction is given by

$$\hat{H}_{ZZ} = \frac{\hbar\zeta}{4}\,\hat{\sigma}_1^z\,\hat{\sigma}_2^z, \tag{24}$$

where $\hbar$ is the reduced Planck constant, $\hat{\sigma}_i^z$ is the Pauli-Z operator acting on qubit i and ζ is the ZZ coupling strength (in angular-frequency units). The only effect of $\hat{H}_{ZZ}$ is to shift the energy of the $|11\rangle$ state relative to the sum of the individual $|10\rangle$ and $|01\rangle$ energies. Equivalently, the transition frequency of qubit 1 depends on whether qubit 2 is in $|0\rangle$ or $|1\rangle$, and vice-versa. The coupling strength ζ is defined operationally by

$$\zeta \;=\; (\omega_{11} - \omega_{10}) \;-\; (\omega_{01} - \omega_{00}), \tag{25}$$

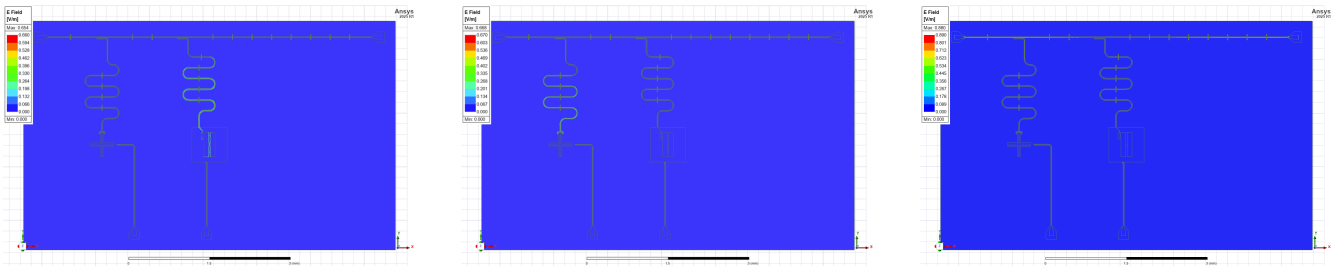
with ω_{ij} the energy of the two-qubit computational state $|ij\rangle$.

Origins: A pure two-level system with the transverse exchange interaction of Eq. (7) would generate iSWAP-type gates but no ZZ term. The ZZ shift emerges because transmons are not two-level systems: their weak anharmonicity α_i leaves the higher levels ($|2\rangle$, $|3\rangle$, ...) close enough in energy that they participate virtually in the dynamics. Second-order perturbation in the coupling g mixes the computational state $|11\rangle$ with the non-computational states $|02\rangle$ and $|20\rangle$, whereas $|01\rangle$ and $|10\rangle$ have no equivalent partner to mix with. The resulting differential energy shift on $|11\rangle$ is the ZZ coupling. In the dispersive regime $|\omega_1 - \omega_2| \gg g$, standard perturbation theory gives [43]

$$\zeta \;\simeq\; -2g^2\left(\frac{1}{\Delta_{12} - \alpha_1} \;+\; \frac{1}{-\Delta_{12} - \alpha_2}\right), \tag{26}$$

with $\Delta_{12} = \omega_1 - \omega_2$. ζ scales with g^2 and vanishes only when $\alpha_i >> \Delta_{12}$

5.2 E-field visualisation

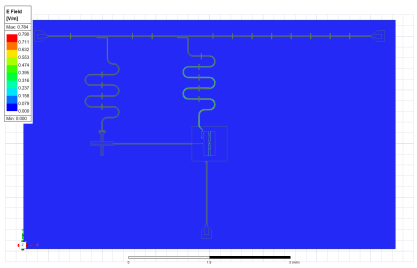


(a) Mode 3 – TQ_1 readout, 6.402 GHz.

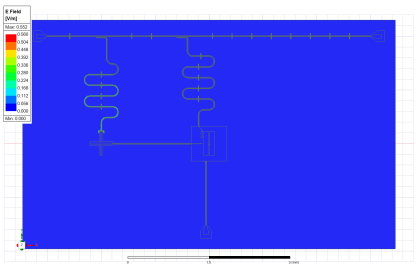
(b) Mode 4 – TQ_2 readout, 6.462 GHz.

(c) Mode 5 – shared feedline, 9.081 GHz.

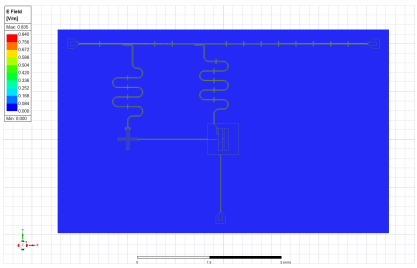
Figure 40: E-field magnitude of the three auxiliary (non-qubit) eigenmodes of the full chip for Model 1 (indirect coupling). The two readout resonators localise their field on the coupled-line tee sections next to their respective qubits (subfigures (a) and (b)); mode 5 is a global mode of the shared feedline connecting the two launch pads (subfigure (c)). Of the three, only mode 3 carries any junction energy — the residual $p_{jj,Q_1} = 0.205$ that signals the Q_1/TQ_1 hybridisation discussed in Sec. 3.1.2 — while modes 4 and 5 carry none, in agreement with Table 4.



(a) Mode 3 – TQ_1 readout, 6.388 GHz.



(b) Mode 4 – TQ_2 readout, 6.453 GHz.



(c) Mode 5 – shared feedline, 9.079 GHz.

Figure 41: Same three auxiliary (non-qubit) eigenmodes for Model 2 (direct coupling). The mode structure is unchanged with respect to Model 1 (Fig. 40), the eigenfrequencies shifting by less than 15 MHz: the extended Xmon arm alters the qubit modes but leaves the readout and feedline modes essentially untouched. As in Model 1, only mode 3 carries a residual junction participation ($p_{jj,Q_1} = 0.205$), in agreement with Table 4.

5.3 LOM implementation

This section describes the by-hand LOM implementation of Sec. 3.2.3. The procedure is identical for both models – only the net name of the Q_2 island differs, and it is detected automatically – and the associated code is reproduced in Annex 5. It proceeds in five steps.

(0) Reduction to the ground-referenced matrix. The extraction counts every potential against the reference of the solver, in which the ground plane is just one more net – the fused conductor identified in Sec. 3.2.2. Its row and column are therefore removed from C_M before anything else, which promotes the ground plane to the reference of the circuit and leaves only the qubit and line nets. The net to remove is recognised automatically through the signature established in Sec. 3.2.2: it is the dominant capacitive partner of every other conductor, and once its contribution is subtracted nothing is left for the others to screen against. The step is not cosmetic: inverting the full matrix instead, i.e. treating the ground plane as one more floating conductor, returns $E_{C,Q_2} = 246$ MHz instead of 173 MHz.

(i) Junction capacitances. The Q3D matrix C_M is purely geometric: the junctions are left as gaps (Sec. 3.2), so the shunt capacitance C_j of each junction must be added by hand. For the floating qubit Q_1, C_j sits between the two pads and enters the matrix as a differential element,

$$C_{tt} \to C_{tt} + C_j, \qquad C_{bb} \to C_{bb} + C_j, \qquad C_{tb} \to C_{tb} - C_j, \tag{27}$$

with $t \equiv$ `pad_top_Q1` and $b \equiv$ `pad_bot_Q1`. For the grounded qubit Q_2, C_j sits between the island and the ground plane and adds to the diagonal alone, $C_{cc} \to C_{cc} + C_j$. We use $C_j = 2$ fF and $L_{j,1} = L_{j,2} = 10$ nH, i.e. the same junction parameters as Burignat, so that the comparison of Tab. 6 is not biased by the junction model.

(ii) Mode vectors and inverse capacitance matrix. The reduced matrix is inverted, $C^{-1} = (C_M + \delta C_j)^{-1}$, and each qubit mode is written as a vector in net space,

$$\mathbf{v}_{Q_1} = \mathbf{e}_t - \mathbf{e}_b, \qquad \mathbf{v}_{Q_2} = \mathbf{e}_c, \tag{28}$$

where $\mathbf{e}_n$ is the unit vector of net n and c denotes the Q_2 island – the cross alone in Model 1, the cross together with its coupler arm in Model 2, the two being galvanically continuous. The difference of the two unit vectors for Q_1 encodes the differential (floating) character of the pocket, the single unit vector for Q_2 its single-ended (grounded) character. No normalisation factor is inserted in $\mathbf{v}_{Q_1}$: the mode variable of a floating transmon is the potential *difference* between its two pads, and for a pair of pads shunted by a capacitance C_S and isolated from everything else one has the exact identity $(\mathbf{e}_t - \mathbf{e}_b)^{\mathsf{T}} C^{-1} (\mathbf{e}_t - \mathbf{e}_b) = 1/C_S$; applied to the full matrix, the same projection returns the screened value. Dividing by $\sqrt{2}$ would halve $(C^{-1})_{11}$ and with it E_{C,Q_1}. Projecting C^{-1} on these vectors gives the effective inverse capacitances

$$(C^{-1})_{ij} = \mathbf{v}_{Q_i}^{\mathsf{T}} \, C^{-1} \, \mathbf{v}_{Q_j}, \qquad i,j \in \{1,2\}. \tag{29}$$

The crucial point is that $(C^{-1})_{11}$ and $(C^{-1})_{22}$ already contain the screening by *every* other conductor of the chip – charge line, feedline and the neighbouring qubit as explicit nets, the readout metal through the grounded reference of step (0) – because the inversion is performed on the whole matrix at once. The stray contributions that a single-qubit treatment would have to enumerate and add by hand are therefore already folded into the projection, and this is what replaces the dressing performed internally by `run_lom()`.

(iii) Transmon parameters. Each qubit is then quantised in the transmon limit $E_J \gg E_C$ [42],

$$\begin{gathered} E_{C,i} = \frac{e^2}{2} (C^{-1})_{ii}, \qquad E_{J,i} = \frac{1}{L_{j,i}} \left(\frac{\Phi_0}{2\pi} \right)^2, \qquad \alpha_i = -E_{C,i}, \\ h f_{\mathrm{lin},i} = \sqrt{8 E_{J,i} E_{C,i}}, \qquad \hbar \omega_{q,i} = h f_{\mathrm{lin},i} - E_{C,i}, \end{gathered} \tag{30}$$

with $\Phi_0 = h/2e$ the flux quantum. The linearised frequency $f_{\mathrm{lin},i}$ is kept separately because it is the quantity the eigenmode branch returns as well, and on which the two branches are compared in Sec. 3.3; the total island capacitance quoted in Tab. 6 follows as $C_{\Sigma,i} = e^2/2E_{C,i} = 1/(C^{-1})_{ii}$. Note that $E_{C,i}$ is obtained here from the projected $(C^{-1})_{ii}$ rather than from $e^2/2C_\Sigma$ evaluated on the bare pads: the two coincide only when the qubit is electrostatically isolated, and the difference is exactly the screening contribution discussed above.

(iv) Qubit–qubit coupling. The transverse coupling between the two qubit modes follows from the off-diagonal projection [58],

$$\frac{g_{12}}{2\pi} = \frac{1}{2} \frac{(C^{-1})_{12}}{\sqrt{(C^{-1})_{11} \, (C^{-1})_{22}}} \sqrt{\omega_{q,1} \, \omega_{q,2}}, \tag{31}$$

which reduces to the familiar $g \simeq \frac{1}{2} \beta \sqrt{\omega_1 \omega_2}$ form with β the capacitance-ratio participation. In Model 1 this term is the residual coupling mediated by the bus and the shared ground; in Model 2 it is dominated by the pad-to-pad capacitance of the extended Xmon arm, which appears in the Q3D matrix as the `pad_top_Q1↔c_connector_arm_Q2` mutual of -5.63 fF (Fig. 36) against -0.018 fF in Model 1 (Fig. 35).

The routine returns E_C, E_J, E_J/E_C, ω_q, α for both qubits and the qubit–qubit coupling g_{12} (Tab. 6).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**