On the Principles of Differentiable Quantum Programming Languages




Shaopeng Zhu   Shih-Han Hung   Shouvanik Chakrabarti   **Xiaodi Wu**

PLDI 2020 - Language Design I

UNIVERSITY OF MARYLAND

JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

A Tale of Two Emerging Programming Languages

Heard of Quantum Programming Languages?

$|0\rangle, |1\rangle, \rho$

$\frac{\partial}{\partial x}[\![P(x)]\!]$

Heard of Differentiable Programming Languages?

This talk is about the happy marriage of both:

Differentiable Quantum Programming Languages

However, it is not just a brain teaser but with strong practical motivation!

With the establishment of **Quantum Supremacy**,

IBM will soon launch a 53-qubit quantum computer

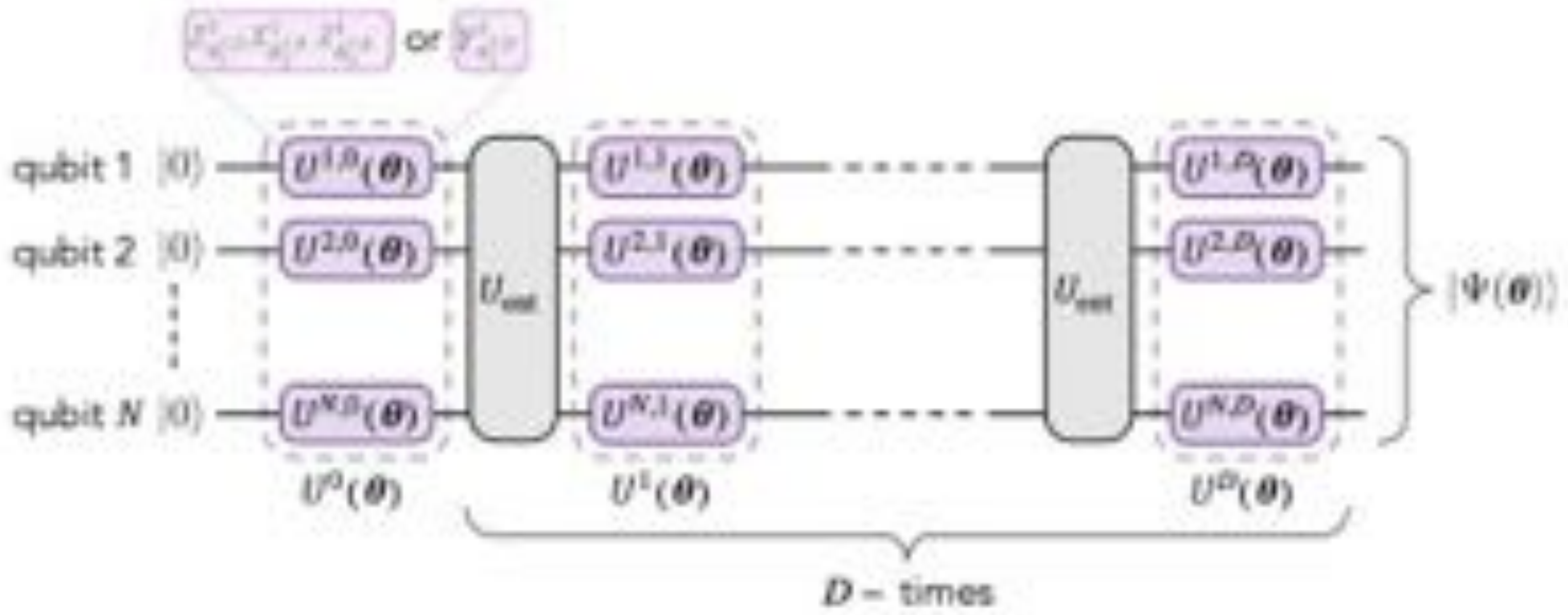
Frederic Lardinois @frederic / 8:00 am EDT • September 18, 2019

Google has reached quantum supremacy – here's what it should do next

TECHNOLOGY | ANALYSIS 26 September 2019

By Chelsea Whyte

Variational Quantum Circuits (VQC)



Its training requires gradient-computation

Infeasible for classical computation power

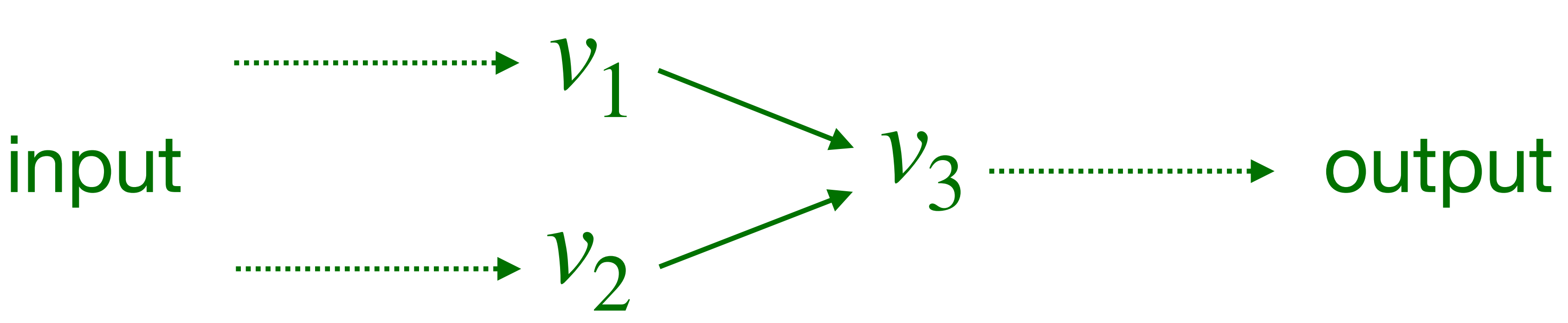
Thus,

using quantum programs to compute the gradients of quantum programs

Is critical for the scalability of gradient-based quantum applications!

Classical vs Quantum 101

classical:



e.g. $M \equiv v_3 = v_1 \times v_2$

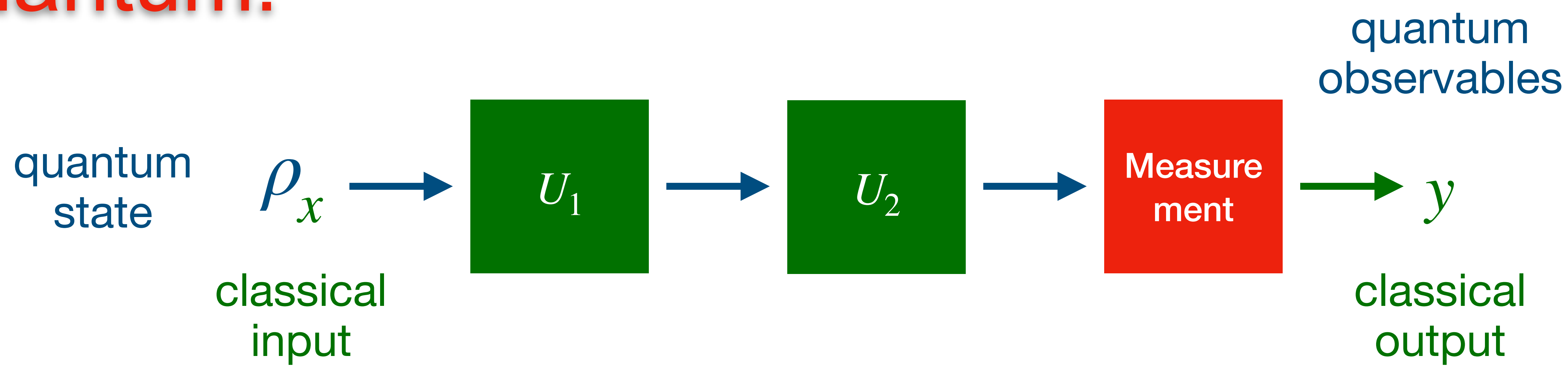
$$\frac{\partial}{\partial \theta} M \equiv v_3 = v_1 \times v_2;$$

$$\dot{v_3} = \dot{v_1} \times v_2 + v_1 \times \dot{v_2}$$

observations:

- actual state (v_i) = representation (v_i)
- all v_i are reals, thus *differentiable*
- store $v_1, v_2, v_3, \dot{v_1}, \dot{v_2}, \dot{v_3}$ at the same time
- **chain-rule**: $\frac{\partial v_3}{\partial \theta} = \frac{\partial v_3}{\partial v_1}\frac{\partial v_1}{\partial \theta} + \frac{\partial v_3}{\partial v_2}\frac{\partial v_2}{\partial \theta}$

quantum:



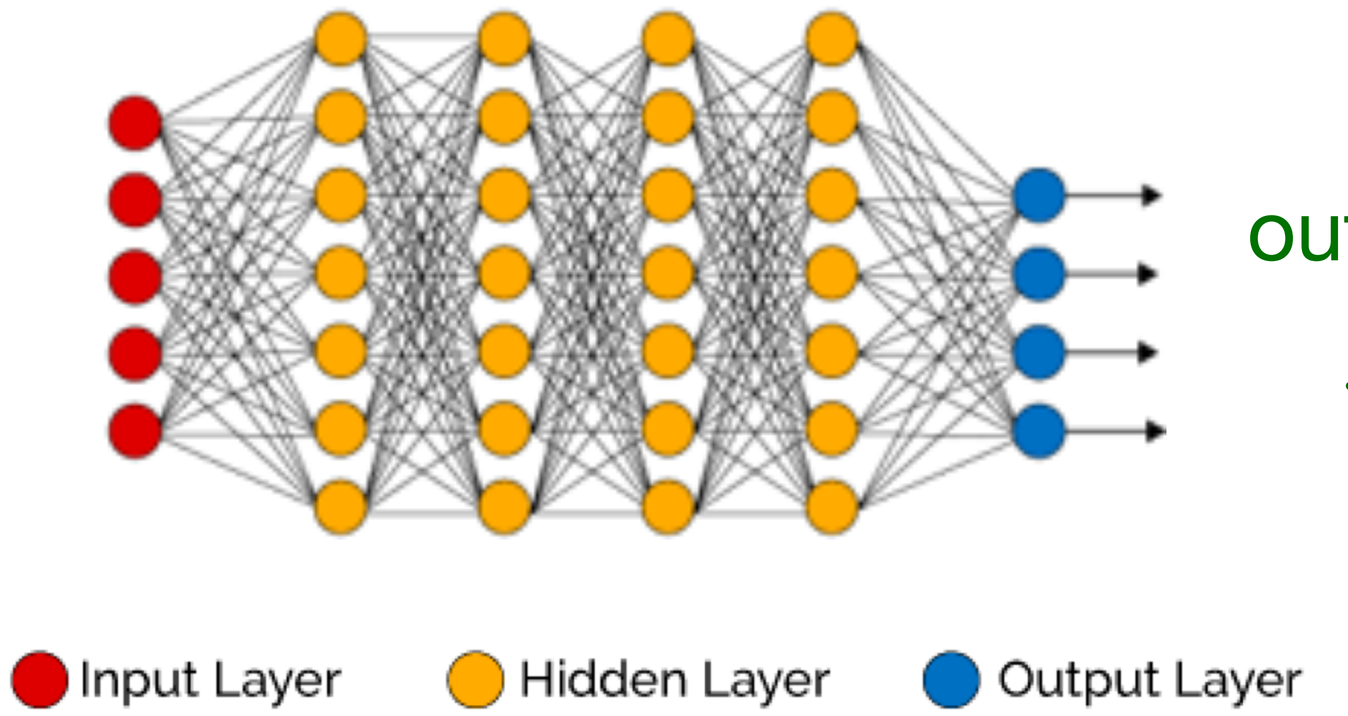
key differences:

- actual state ρ is a quantum state; its classical representation is an exponential (in # of qbits) matrix.
- Unit operation U_1 takes 1 unit time on q. machines; classically simulating $U_1 \rho U_1^\dagger$ takes exponential time.
- a priori unclear $\frac{\partial \rho}{\partial \theta}$ for both ρ and θ part
- cannot store all intermediate ρ due to **no-cloning**
- hard to make sense of chain-rules

Classical vs Quantum 101: cont'd

Classical Neural Networks (CNNs)

input
x



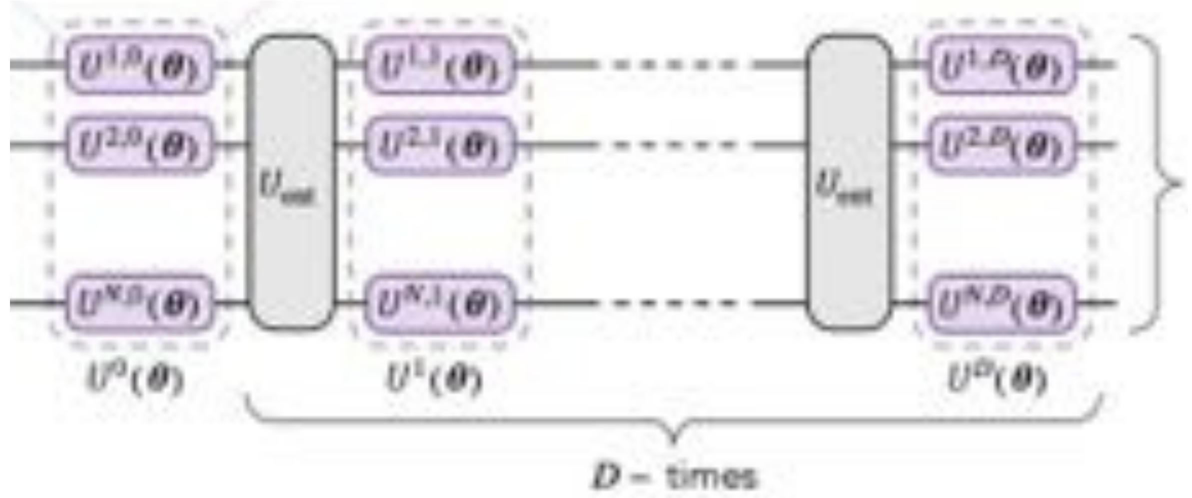
output
y

Replace $x \to y$ (classical) by

$x \to \rho_x \to y$ (quantum) w/ potential speedups

input
x

quantum
ρ_x

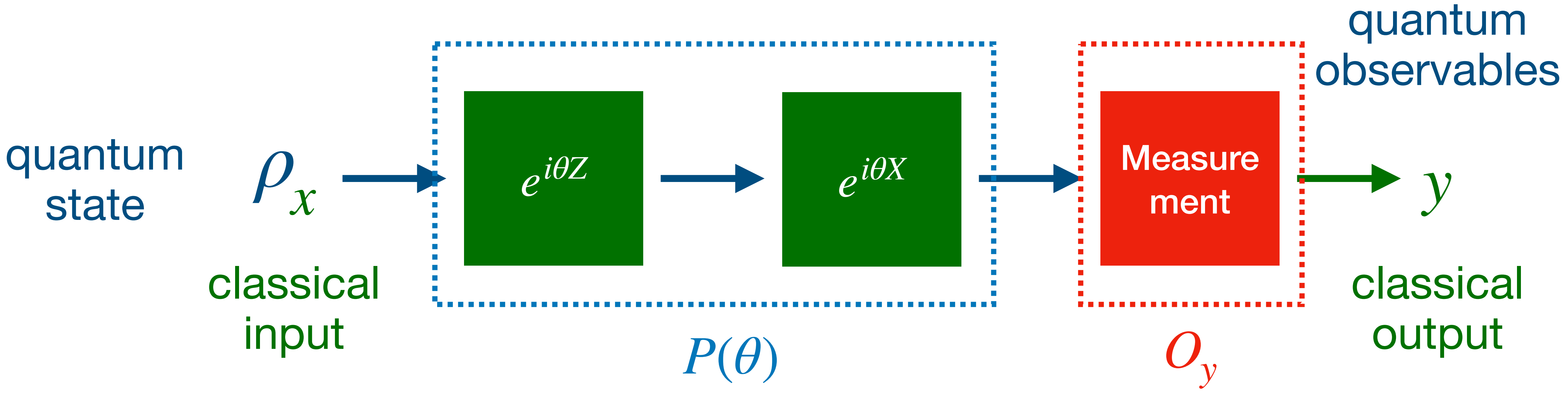


output
y

Measure
ment

Variational Quantum Circuits (VQCs)

Variational Quantum Circuits (VQCs)



Training of VQCs, similar to CNNs, will optimize the **loss** functions

$\text{loss} = L(x, y, \theta)$ for VQC $P(\theta)$ w/ d. semantics $[\![P(\theta)]\!]$

$$y(x,\theta) = \mathrm{Tr}(O_y[\![P(\theta)]\!](\rho_x))$$

quantum
state

quantum
observable

Gradient $\dfrac{\partial L}{\partial \theta}$ can be computed from $\dfrac{\partial y}{\partial \theta}$ exponential classical cost

$\exists$ q. gadget for simple P *w/o* **formal formulation** and **program features**

Conceptual challenges from the beginning and more to come!

Contributions

"Deep Learning est mort. Vive Differentiable Programming!"

----- Yann LeCun

- **Formal Formulation** of Differentiable Quantum Programming Languages:
    - *basic concepts*: **parameterized quantum programs, semantics of differentiation**
    - *code-transformation*: **two-stage code-transformation, a logic proving its correctness**
    - *features*: **support controls, compositions, also w/ resource efficiency**
- **Implementation** of a prototype in OCaml and benchmark tests on representative cases
    - *quantum neuro-symbolic application*: **parameterized quantum programs in machine learning**
    - *resource efficiency*: **empirically demonstrated efficiency for representative cases**

Differentiable Quantum Prog-Lang:
github:/LibertasSpZ/adcompile

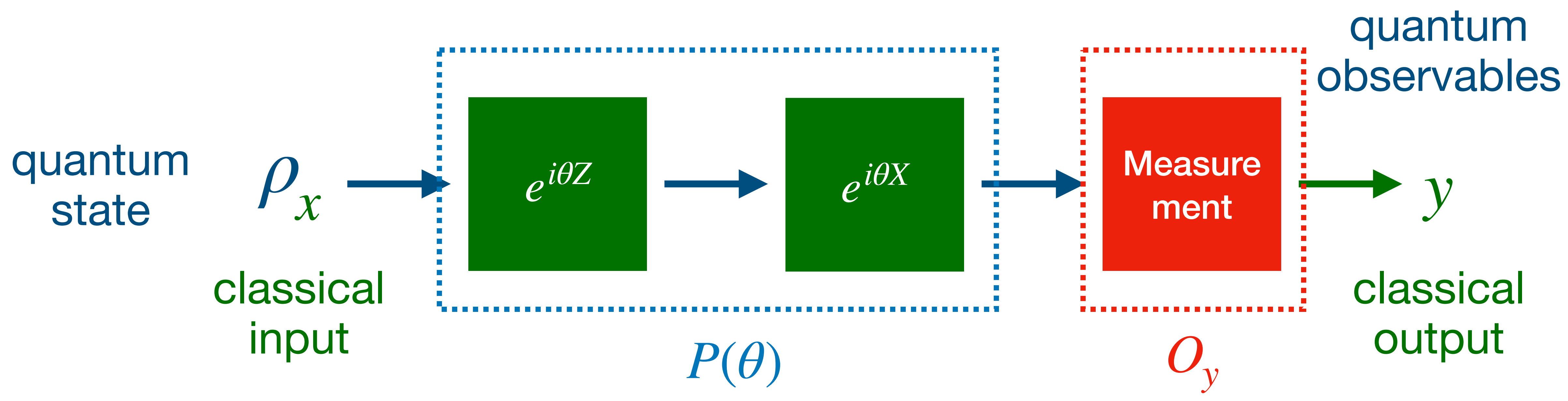
Formal Formulations:

- Use **parameterized quantum while-language** to formulate $P(\theta)$ with *classically parameterized* Pauli rotation gates: e.g.,

$$e^{i\theta Z}, e^{i\theta X \otimes X}$$

which forms a *universal* gate set and can be readily implemented on near-term quantum machines.

Variational Quantum Circuits (VQCs)



- Model quantum observable on $P(\theta)$ by **Observable Semantics**

$$[\![(O_y, \rho_x) \rightarrow P(\theta)]\!] \equiv \mathrm{Tr}(O_y [\![P(\theta)]\!](\rho_x))$$

matches exactly the observable quantum output

Note that it will serve as (1) the target to differentiate; and (2) the read-out of any quantum programs.

- Make sense of $\frac{\partial}{\partial \theta} P(\theta)$ computing the derivative of $P(\theta)$ (**Differential Semantics**)

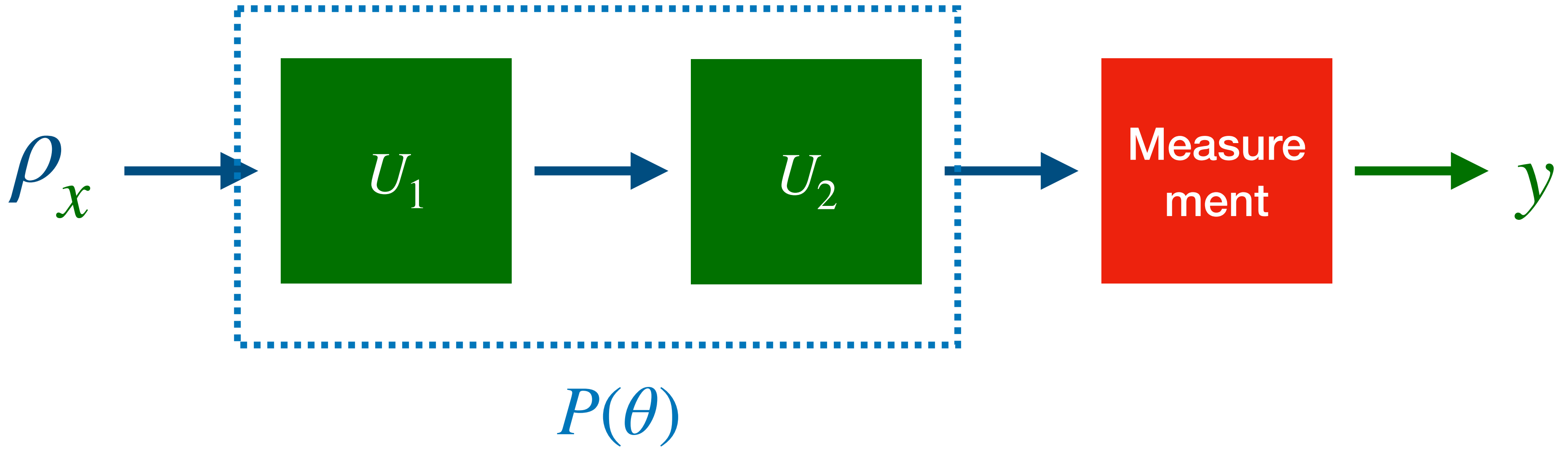
$$[\![(O_y, \rho_x) \rightarrow \frac{\partial}{\partial \theta} P(\theta)]\!] = \frac{\partial}{\partial \theta} [\![(O_y, \rho_x) \rightarrow P(\theta)]\!]$$

one $\frac{\partial}{\partial \theta} P(\theta)$ for any O_y and ρ_x

strong requirement: achievable and critical

Formal Formulations: cont'd

Move to code-transformation and construction of $\frac{\partial}{\partial\theta}P(\theta)$ based on previous formulations



$$P(\theta) \equiv U_1(\theta); U_2(\theta);$$

$$\frac{\partial}{\partial\theta}M \equiv v_3 = v_1 \times v_2;$$
$$\dot{v_3} = \dot{v_1} \times v_2 + v_1 \times \dot{v_2}$$

classical analogue

$$\frac{\partial}{\partial\theta}P(\theta) \equiv \frac{\partial}{\partial\theta}U_1(\theta); U_2(\theta); + U_1(\theta); \frac{\partial}{\partial\theta}U_2(\theta); \quad \textbf{??}$$

Making sense of "+":

classical "+": run both $\dot{v_1} \times v_2$ and $v_1 \times \dot{v_2}$ on the **input**, and then sum the outputs

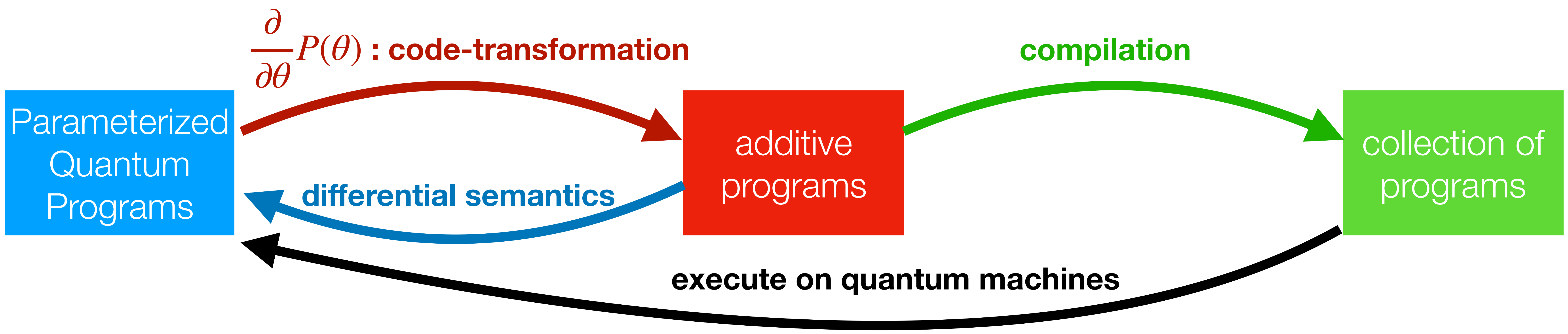
quantum "+": hope to do the same. However, the **input cannot be cloned**, and will be **consumed** for each

Thus, $\frac{\partial}{\partial\theta}P(\theta)$ needs to be a **collection** of programs running on **copies** of the input state —> **complication**!

Ideally, (1) hope to have a **similar code-transformation like classical** for intuition and implementation, but still able to keep track of the right collection of programs in an efficient way.

(2) hope to control the **size** of the collection —> **# of copies** of the input state for efficiency.

Two-stage Code-Transformation



additive programs

Quantum While-Programs + Sum Operator

$$\begin{aligned} \underline{P(\boldsymbol{\theta})} \;::=\; & \underline{\mathbf{abort}[\overline{q}]} \mid \underline{\mathbf{skip}[\overline{q}]} \mid \underline{q := |0\rangle} \mid \underline{\overline{q} := U(\boldsymbol{\theta})[\overline{q}]} \mid \\ & \underline{P_1(\boldsymbol{\theta})}; \underline{P_2(\boldsymbol{\theta})} \mid \underline{\mathbf{case}}\ M[\overline{q}] = \overline{m \rightarrow \underline{P_m(\boldsymbol{\theta})}}\ \mathbf{end} \mid \\ & \underline{\mathbf{while}^{(T)}}\ M[\overline{q}] = 1\ \mathbf{do}\ \underline{P_1}(\boldsymbol{\theta})\ \mathbf{done} \mid \boxed{\underline{P_1(\boldsymbol{\theta})} + \underline{P_2(\boldsymbol{\theta})}} \end{aligned}$$

Operational Semantics of Sum

$$\langle \underline{P_1(\boldsymbol{\theta}^*)} + \underline{P_2(\boldsymbol{\theta}^*)}, \rho\rangle \rightarrow \langle \underline{P_1(\boldsymbol{\theta}^*)}, \rho\rangle,$$
$$\langle \underline{P_1(\boldsymbol{\theta}^*)} + \underline{P_2(\boldsymbol{\theta}^*)}, \rho\rangle \rightarrow \langle \underline{P_2(\boldsymbol{\theta}^*)}, \rho\rangle$$

exactly match the intuition of "+" for differential semantics

compilation

Output a collection of quantum programs while keeping the size small

$$\frac{\partial}{\partial\theta}P(\theta) \equiv \frac{\partial}{\partial\theta}U_1(\theta); U_2(\theta); + U_1(\theta); \frac{\partial}{\partial\theta}U_2(\theta);$$

$$\text{Compile}(\frac{\partial}{\partial\theta}P(\theta)) \equiv \{| \frac{\partial}{\partial\theta}U_1(\theta); U_2(\theta); , U_1(\theta); \frac{\partial}{\partial\theta}U_2(\theta); |\}, \text{ size } 2$$

general compilation follows the same intuition but more complicated :

(Atomic) $\texttt{Compile}(P(\boldsymbol{\theta})) \equiv \{|P(\boldsymbol{\theta})|\}$,
if $P(\boldsymbol{\theta}) \equiv \underline{\mathbf{abort}[\overline{v}]} \mid \underline{\mathbf{skip}[\overline{v}]} \mid \underline{q := |0\rangle}$
$\mid \underline{\overline{v} := U(\boldsymbol{\theta})[\overline{v}]}$.

(Sequence) $\texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}; \underline{P_2(\boldsymbol{\theta})}) \equiv$
$$\begin{cases} \{|\mathbf{abort}|\}, \text{ if } \texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}) = \{|\mathbf{abort}|\}; \\ \{|\mathbf{abort}|\}, \text{ if } \texttt{Compile}(\underline{P_2(\boldsymbol{\theta})}) = \{|\mathbf{abort}|\}; \\ \{|Q_1(\boldsymbol{\theta}); Q_2(\boldsymbol{\theta}) : Q_b(\boldsymbol{\theta}) \in \texttt{Compile}(\underline{P_b(\boldsymbol{\theta})})|\}, \\ \qquad \text{otherwise.} \end{cases}$$

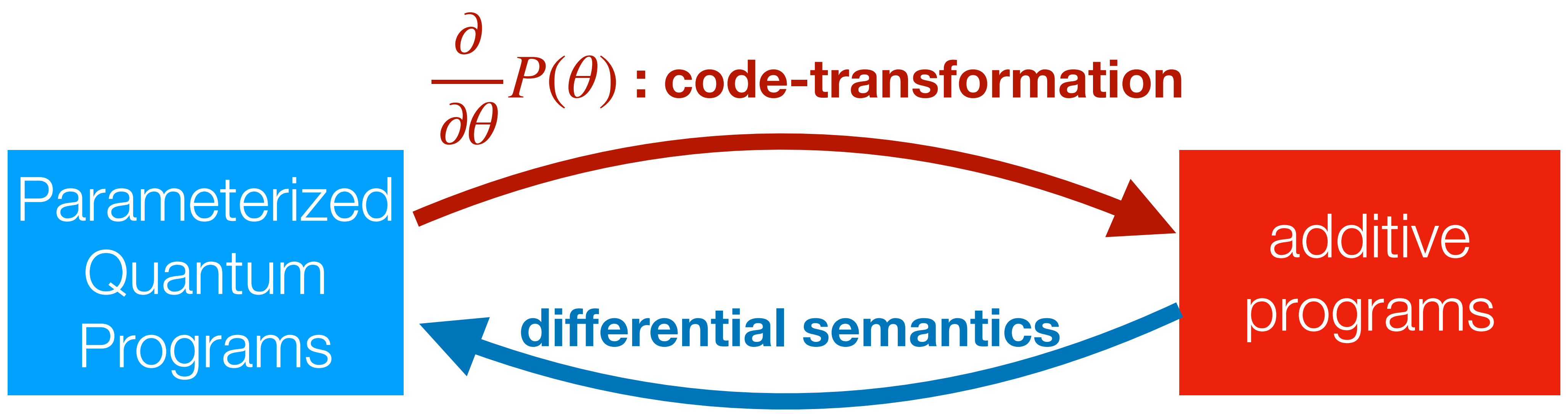
(Case m) $\texttt{Compile}(\underline{\mathbf{case}}) \equiv \text{FB}(\underline{\mathbf{case}})$, described in Fig.3b.

(While$^{(T)}$) $\texttt{Compile}(\underline{\mathbf{while}^{(T)}})$: use (Case m) and (Sequence).

(Sum) $\texttt{Compile}(\underline{P_1(\boldsymbol{\theta})} + \underline{P_2(\boldsymbol{\theta})}) \equiv$
$$\begin{cases} \texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}) \biguplus \texttt{Compile}(\underline{P_2(\boldsymbol{\theta})}), \text{ if } \forall b \in \{1, \\ \qquad 2\}, \texttt{Compile}(\underline{P_b(\boldsymbol{\theta})}) \neq \{|\mathbf{abort}|\}; \\ \texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}), \text{ if } \texttt{Compile}(\underline{P_2(\boldsymbol{\theta})}) = \{|\mathbf{abort}|\}, \\ \qquad \texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}) \neq \{|\mathbf{abort}|\}; \\ \texttt{Compile}(\underline{P_2(\boldsymbol{\theta})}), \text{ if } \texttt{Compile}(\underline{P_1(\boldsymbol{\theta})}) = \{|\mathbf{abort}|\}, \\ \qquad \texttt{Compile}(\underline{P_2(\boldsymbol{\theta})}) \neq \{|\mathbf{abort}|\}; \\ \{|\mathbf{abort}|\}, \text{otherwise} \end{cases}$$

Code-Transformation

(Trivial)	$\frac{\partial}{\partial\theta}(\mathbf{abort}[\bar{v}]), \frac{\partial}{\partial\theta}(\mathbf{skip}[\bar{v}]), \frac{\partial}{\partial\theta}(q := \lvert 0\rangle) \equiv \mathbf{abort}[\bar{v} \cup \{A\}]$.
(Trivial-U)	$\frac{\partial}{\partial\theta}(\bar{v} := U(\boldsymbol{\theta})[\bar{v}]) \equiv \mathbf{abort}[\bar{v} \cup \{A\}]$, if $\theta_j \notin \boldsymbol{\theta}$.
(1-qb)	$\frac{\partial}{\partial\theta}(q_1 := R_\sigma(\theta)[q_1]) \equiv A, q_1 := R'_\sigma(\theta)[A, q_1]$.
(2-qb)	$\frac{\partial}{\partial\theta}(q_1, q_2 := R_{\sigma\otimes\sigma}(\theta)[q_1, q_2]) \equiv A, q_1, q_2 := R'_{\sigma\otimes\sigma}(\theta)[A, q_1, q_2]$.
(Sequence)	$\frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta}); S_2(\boldsymbol{\theta})) \equiv (S_1(\boldsymbol{\theta}); \frac{\partial}{\partial\theta}(S_2(\boldsymbol{\theta}))) + (\frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta})); S_2(\boldsymbol{\theta}))$.
(Case)	$\frac{\partial}{\partial\theta}(\mathbf{case}\ M[\bar{q}] = m \rightarrow S_m(\boldsymbol{\theta})\ \mathbf{end}) \equiv \mathbf{case}\ M[\bar{q}] = m \rightarrow \frac{\partial}{\partial\theta}(S_m(\boldsymbol{\theta}))\ \mathbf{end}$.
(while$^{(T)}$)	Use (Case) and (Sequence).
(S-C)	$\frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta}) + S_2(\boldsymbol{\theta})) \equiv \frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta})) + \frac{\partial}{\partial\theta}(S_2(\boldsymbol{\theta}))$.



We develop a sound logic to prove its correctness.

Similar to classical for the convenience of compiler implementation!

(1) modify existing *phase-shift* rule using two-circuit-difference to one circuit with super-posed control for composition and efficiency.

(2) the proof relies on the **strong** requirement in differential semantics

$$[\![(O,\rho) \rightarrow \frac{\partial}{\partial\theta}(S_0(\boldsymbol{\theta}); S_1(\boldsymbol{\theta}))]\!] = [\![(O,\rho) \rightarrow \frac{\partial}{\partial\theta}(S_0(\boldsymbol{\theta})); S_1(\boldsymbol{\theta})]\!] + [\![(O,\rho) \rightarrow S_0(\boldsymbol{\theta}); \frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta}))]\!].$$

$$[\![(O,\rho) \rightarrow S_0(\boldsymbol{\theta}); \frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta}))]\!] = [\![\boxed{(O, [\![S_0(\boldsymbol{\theta})]\!](\rho))} \rightarrow \frac{\partial}{\partial\theta}(S_1(\boldsymbol{\theta}))]\!]$$

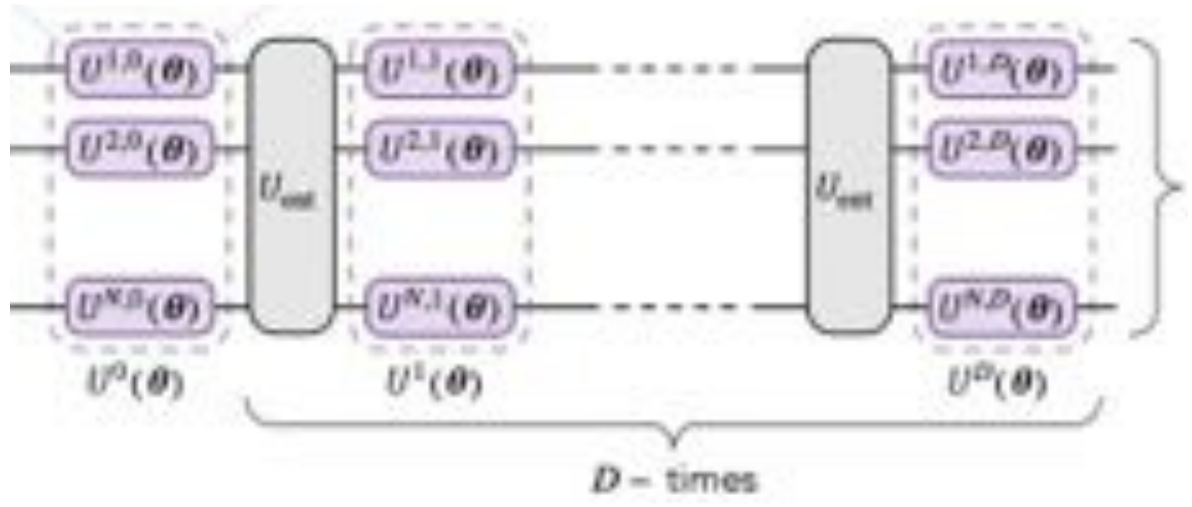
$$[\![(O,\rho) \rightarrow \frac{\partial}{\partial\theta}(S_0(\boldsymbol{\theta})); S_1(\boldsymbol{\theta})]\!] = [\![\boxed{([\![S_1(\boldsymbol{\theta})]\!]^*(O), \rho)} \rightarrow \frac{\partial}{\partial\theta}(S_0(\boldsymbol{\theta}))]\!]$$

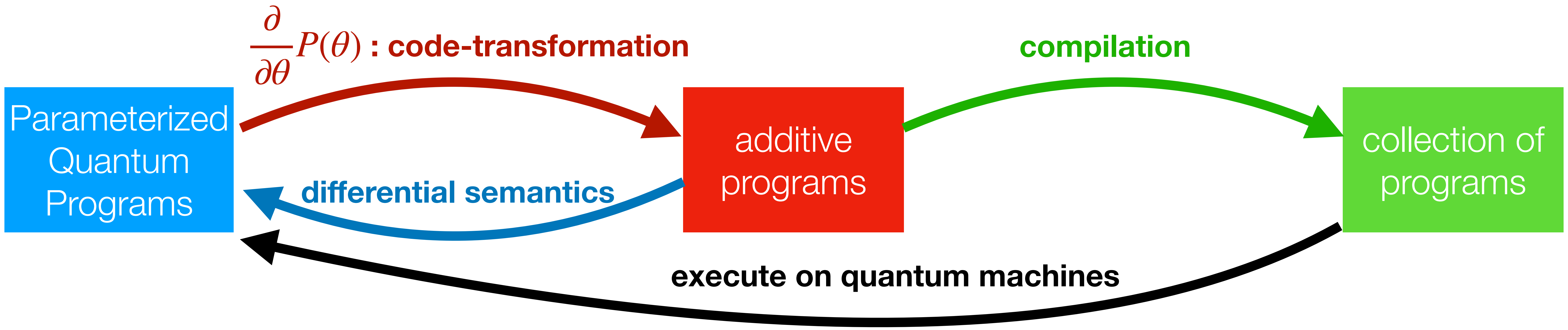
make use of the premises of $\frac{\partial}{\partial\theta}S_0(\theta)$, $\frac{\partial}{\partial\theta}S_1(\theta)$, for **different** (O, ρ) pairs

(3) support **case** unconditionally, better than the classical case

support **bounded loop**, matching the classical case [Plotkin, POPL'18]

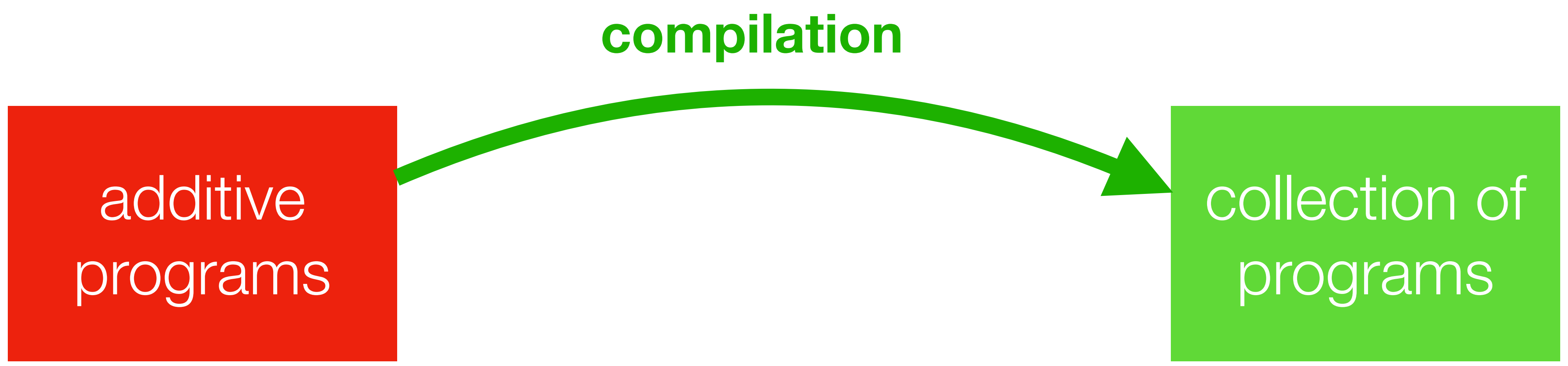
Recap …





- **Formal Formulation** of Differentiable Quantum Programming Languages:
  - *basic concepts*: **parameterized quantum programs, semantics of differentiation**
  - *code-transformation*: **two-stage code-transformation, a logic proving its correctness**
  - *features*: **support controls, compositions, also w/ resource efficiency**
- **Implementation** of a prototype in OCaml and benchmark tests on representative cases
  - *quantum neuro-symbolic application*: **parameterized quantum programs in machine learning**
  - *resource efficiency*: **empirically demonstrated efficiency for representative cases**

Resource Estimation



Definition 7.1. *The "Occurrence Count for θ_j" in $P(\boldsymbol{\theta})$, denoted $\mathrm{OC}_j(P(\boldsymbol{\theta}))$, is defined as follows:*

1. *If* $P(\boldsymbol{\theta}) \equiv \mathbf{abort}[\overline{v}] | \mathbf{skip}[\overline{v}] | q := |0\rangle$ $(q \in \overline{v})$, *then* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = 0$;
2. $P(\boldsymbol{\theta}) \equiv U(\boldsymbol{\theta})$: *if* $U(\boldsymbol{\theta})$ *trivially uses* θ_j, *then* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = 0$; *otherwise* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = 1$.
3. *If* $P(\boldsymbol{\theta}) \equiv U(\boldsymbol{\theta}) = P_1(\boldsymbol{\theta}); P_2(\boldsymbol{\theta}))$ *then* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = \mathrm{OC}_j(P_1(\boldsymbol{\theta})) + \mathrm{OC}_j(P_2(\boldsymbol{\theta}))$.
4. *If* $P(\boldsymbol{\theta}) \equiv \mathbf{case}\ M[\overline{q}] = \overline{m \rightarrow P_m(\boldsymbol{\theta})}\ \mathbf{end}$ *then* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = \max_m \mathrm{OC}_j(P_m(\boldsymbol{\theta}))$.
5. *If* $P(\boldsymbol{\theta}) \equiv \mathbf{while}^{(T)}\ M[\overline{q}] = 1\ \mathbf{do}\ P_1(\boldsymbol{\theta})\ \mathbf{done}$ *then* $\mathrm{OC}_j(P(\boldsymbol{\theta})) = T \cdot \mathrm{OC}_j(P_1(\boldsymbol{\theta}))$.

Generally **EXP** (in # of "+"s) programs after compilation

However, for relevant $\frac{\partial}{\partial \theta}P(\theta)$, we show the # is **bounded** by **occurrence count** of θ

which basically counts the # of appearances of θ

$$P(\theta) \equiv U_1(\theta); U_2(\theta);$$ occurrence count 2

Compile($\frac{\partial}{\partial \theta}P(\theta)$) $\equiv \{| \frac{\partial}{\partial \theta}U_1(\theta); U_2(\theta); , U_1(\theta); \frac{\partial}{\partial \theta}U_2(\theta); |\}$, size 2

Why occurrence could be a reasonable quantity?

$v_3 = v_1 \times v_2;$ occurrence count 2: $v_1(\theta), v_2(\theta)$ in v_3

$\dot{v_3} = \dot{v_1} \times v_2 + v_1 \times \dot{v_2}$ instead of extra initial states, classically one needs 2 extra registers to store $\dot{v_1}, \dot{v_2}$

$\geq$

$P(\boldsymbol{\theta})$	$\mathrm{OC}(\cdot)$	$\lvert\#\frac{\partial}{\partial\theta}(\cdot)\rvert$	#gates	#lines	#layers	#qb
$\mathrm{QNN}_{M,i}$	24	24	165	189	3	18
$\mathrm{QNN}_{M,w}$	56	24	231	121	5	18
$\mathrm{QNN}_{L,i}$	48	48	363	414	6	36
$\mathrm{QNN}_{L,w}$	504	48	2079	244	33	36
$\mathrm{VQE}_{M,i}$	15	15	224	241	3	12
$\mathrm{VQE}_{M,w}$	35	15	224	112	5	12
$\mathrm{VQE}_{L,i}$	40	40	576	628	5	40
$\mathrm{VQE}_{L,w}$	248	40	1984	368	17	40
$\mathrm{QAOA}_{M,i}$	18	18	120	142	3	18
$\mathrm{QAOA}_{M,w}$	42	18	168	94	5	18
$\mathrm{QAOA}_{L,i}$	36	36	264	315	6	36
$\mathrm{QAOA}_{L,w}$	378	36	1512	190	33	36

Benchmark tests on **typical** instances

Quantum Neuro-Symbolic Application

"Deep Learning est mort. Vive Differentiable Programming!"

----- Yann LeCun

$$\begin{aligned} Q(\Gamma) \equiv \quad & R_X(\gamma_1)[q_1]; R_X(\gamma_2)[q_2]; R_X(\gamma_3)[q_3]; R_X(\gamma_4)[q_4]; \\ & R_Y(\gamma_5)[q_1]; R_Y(\gamma_6)[q_2]; R_Y(\gamma_7)[q_3]; R_Y(\gamma_8)[q_4]; \\ & R_Z(\gamma_9)[q_1]; R_Z(\gamma_{10})[q_2]; R_Z(\gamma_{11})[q_3]; R_Z(\gamma_{12})[q_4], \end{aligned}$$

(no control) $P_1(\Theta, \Phi) \equiv Q(\Theta); Q(\Phi).$

(w/ control) $P_2(\Theta, \Phi, \Psi) \equiv Q(\Theta); \textbf{case } M[q_1] = 0 \rightarrow \quad Q(\Phi)$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad 1 \rightarrow \quad Q(\Psi).$

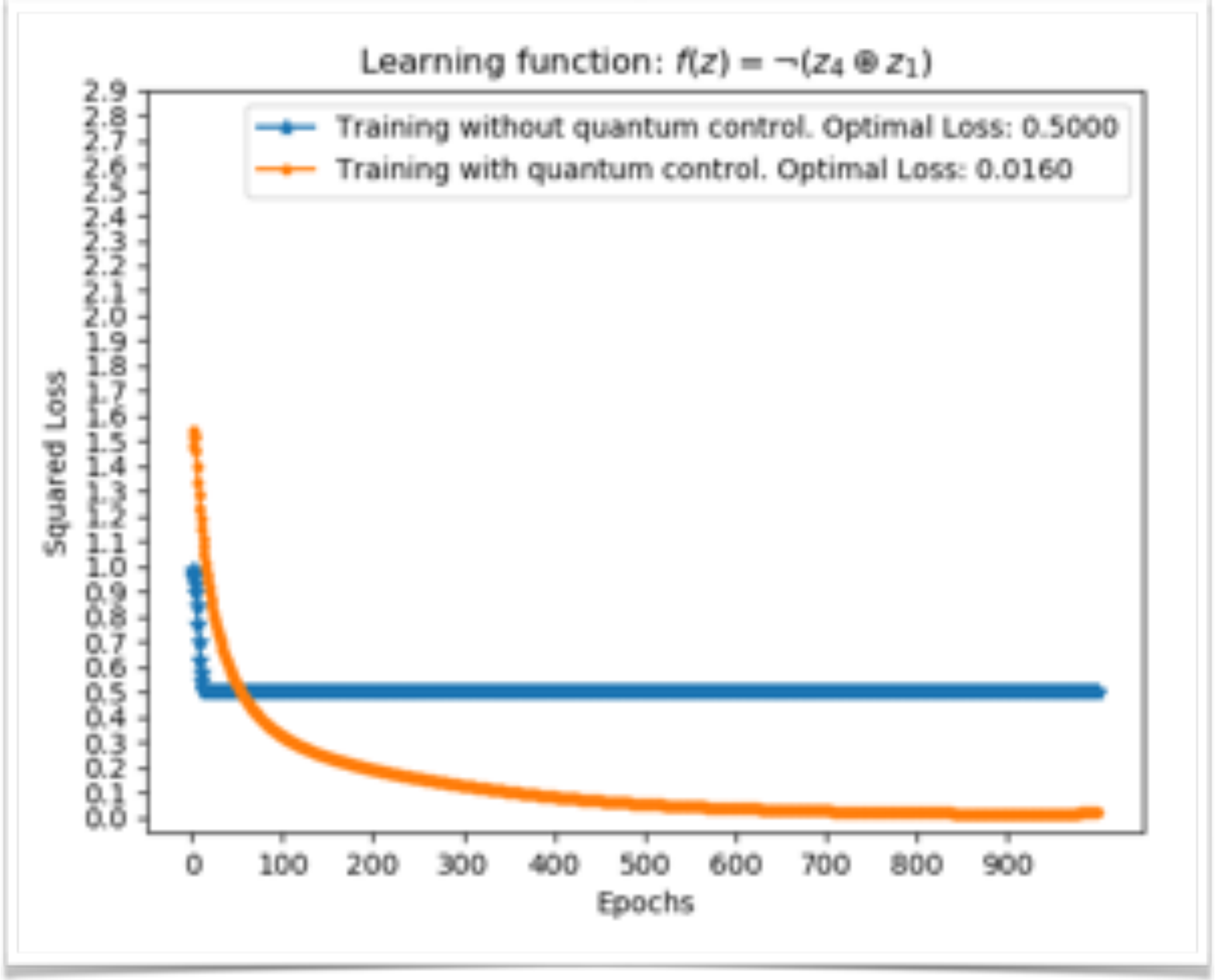
Note that $P_1(\Theta, \Phi)$ and $P_2(\Theta, \Phi, \Psi)$ run the same # of gates.

Simple classification task w/ ground truth

$$f(z) = \neg(z_1 \oplus z_4), z = z_1 z_2 z_3 z_4 \in \{0,1\}^4$$

via the following **square loss** function

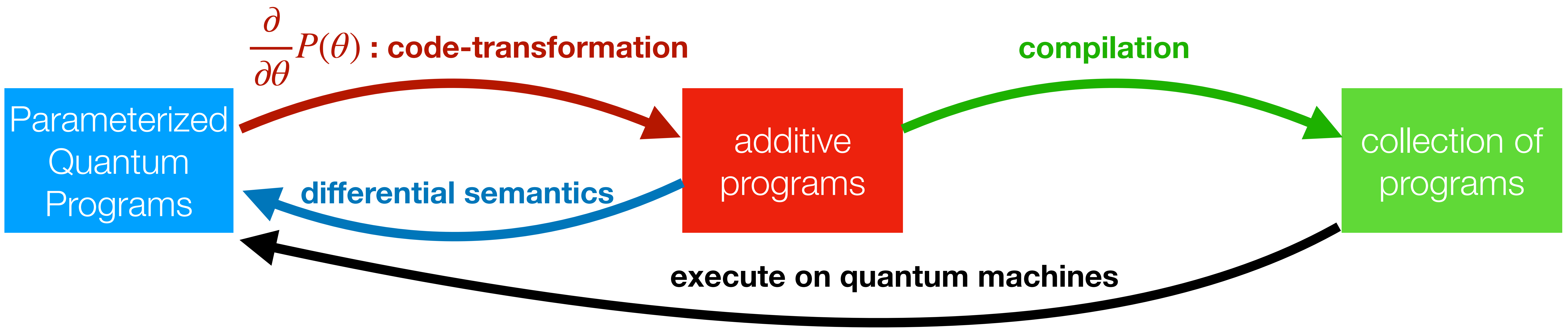
$$\text{loss} = \sum_{z \in \{0,1\}^4} 0.5 * (l_{\boldsymbol{\theta}}(z) - f(z))^2$$



Training of P_2 is conducted on our implementation, whereas the training of P_1 is conducted on prior art (e.g., Pennylane).

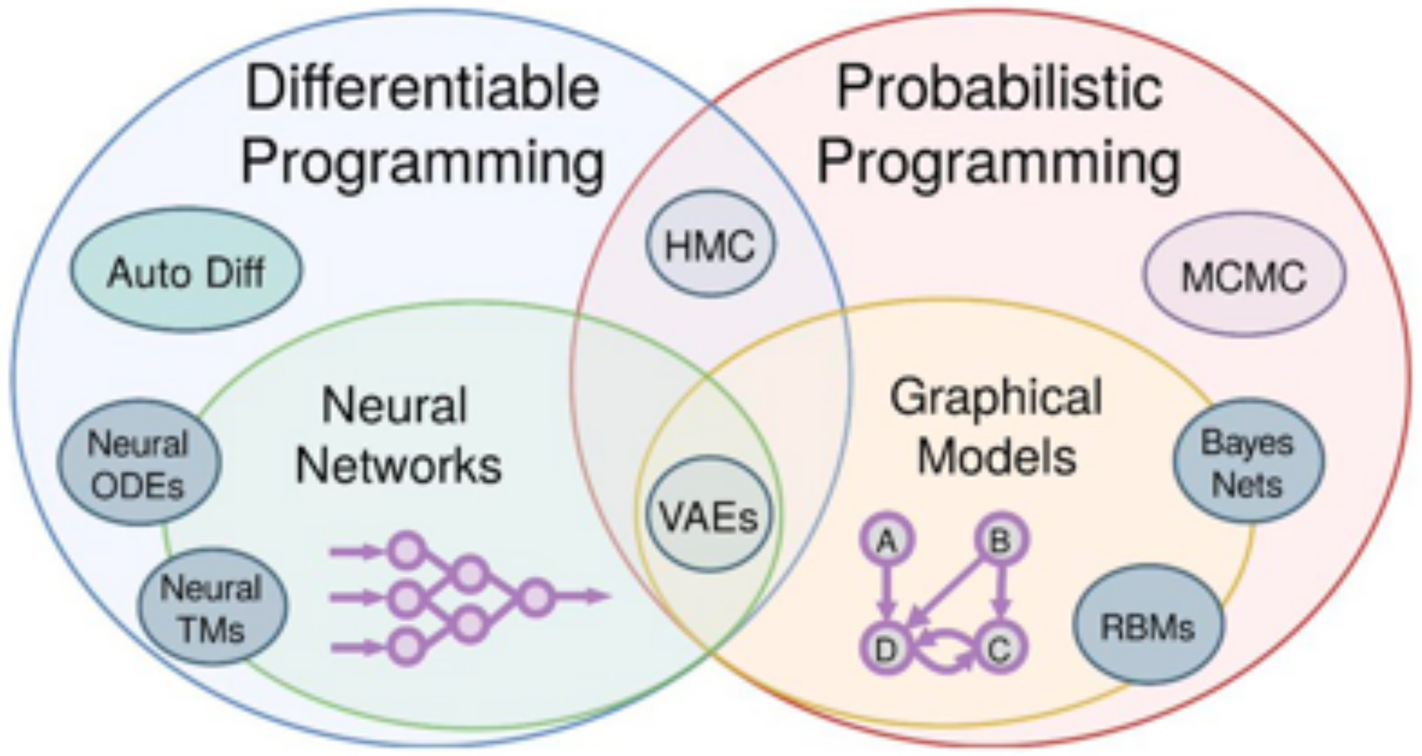
The use of quantum control could be **significant** for near-term.

Conclusions



open questions:
functional PL,
q. neuro-symbolic apps

even if you are only interested in classical



As a generalization of both,
our findings might provide hints of unifying differentiable and probabilistic programming!

THANK YOU!

Differentiable Quantum Prog-Lang:
github:/LibertasSpZ/adcompile