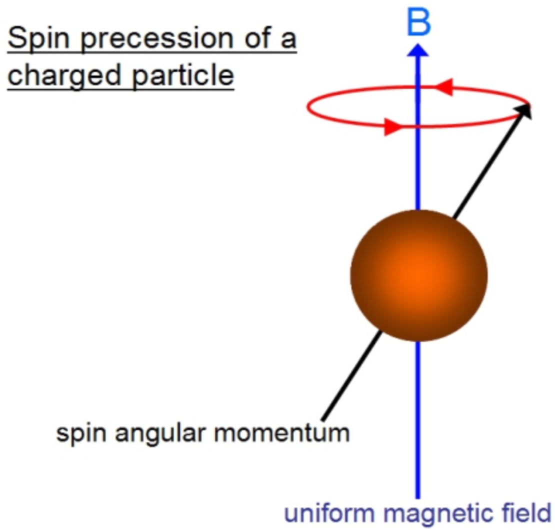
A Surrogate Model to Optimize Injection Efficiency in PSI muEDM Experiment

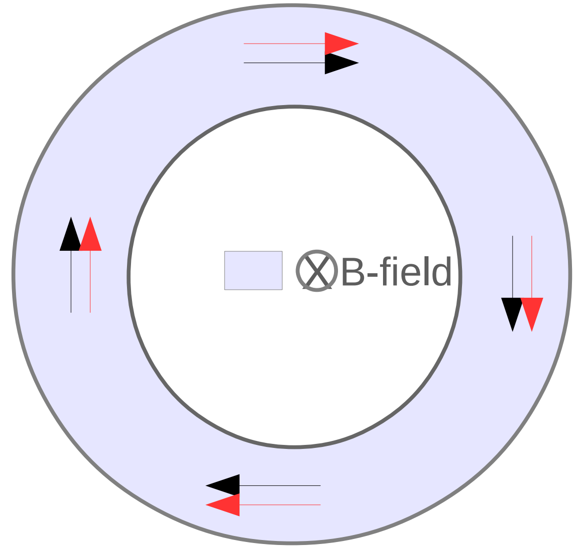
Ritwika Chakraborty (PSI)

European AI for Fundamental Physics Conference
Amsterdam

02.05.2024

Muons in a Storage Ring

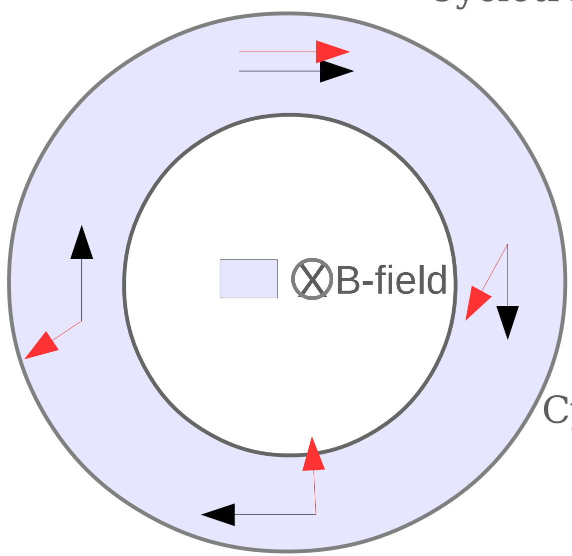




g = 2

Cyclotron = spin precession

In presence of vacuum effects:



g > 2

Cyclotron ≠ spin precession

Muons Electric Dipole Moment (EDM)

In general, relativistic muons, in presence of electric fields + magnetic field

$$\vec{\Omega} = \underbrace{\vec{\Omega}_0}_{\text{Spin precession}} - \underbrace{\vec{\Omega}_c}_{\text{Cyclotron}}$$

Thomas-BMT equation for spin dynamics in EM fields:

$$\vec{\Omega} = \underbrace{\frac{q}{m}\left[a\vec{B} - \frac{a\gamma}{(\gamma+1)}\left(\vec{\beta}\cdot\vec{B}\right)\vec{\beta} - \left(a + \frac{1}{1-\gamma^2}\right)\frac{\vec{\beta}\times\vec{E}}{c}\right]}_{\text{g-2 term}} + \underbrace{\frac{\eta q}{2m}\left[\vec{\beta}\times\vec{B} + \frac{\vec{E}}{c} - \frac{\gamma c}{(\gamma+1)}\left(\vec{\beta}\cdot\vec{E}\right)\vec{\beta}\right]}_{\text{EDM term}}$$

- Non-zero muon EDM indicates CP-violation
- Standard model prediction ~10^{-38} e.cm
- PSI muon EDM sensitivity target 6 x 10^{-23} e.cm → ~3 order of magnitude better than current limit

Frozen Spin Technique

- E ⊥B ⊥ β

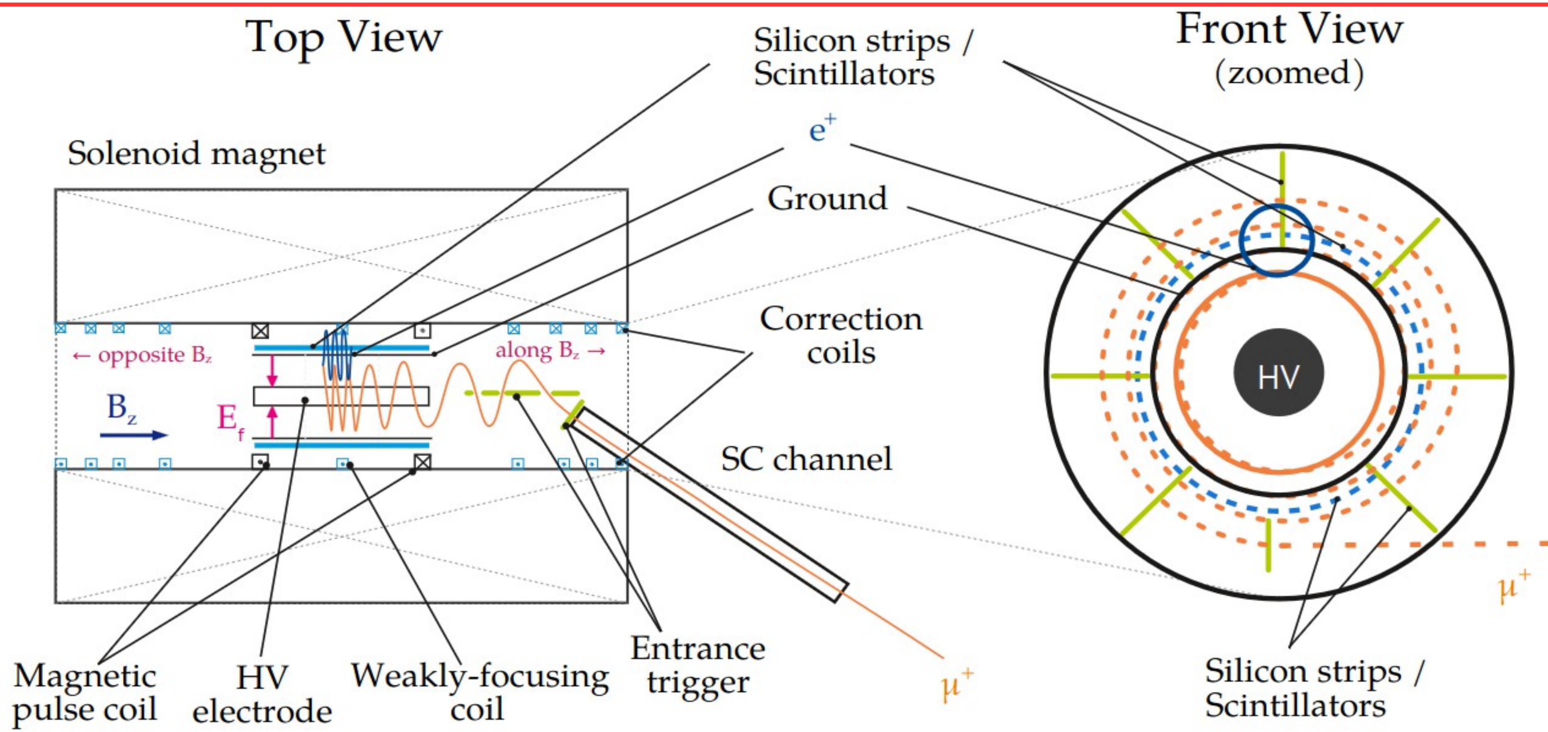
$$\vec{\Omega} = \underbrace{\frac{q}{m}\left[a\vec{B} - \cancel{\frac{a\gamma}{(\gamma+1)}\left(\vec{\beta}\cdot\vec{B}\right)\vec{\beta}} - \left(a + \frac{1}{1-\gamma^2}\right)\frac{\vec{\beta}\times\vec{E}}{c}\right]}_{\text{g-2 term}} + \underbrace{\frac{\eta q}{2m}\left[\vec{\beta}\times\vec{B} + \frac{\vec{E}}{c} - \cancel{\frac{\gamma c}{(\gamma+1)}\left(\vec{\beta}\cdot\vec{E}\right)\vec{\beta}}\right]}_{\text{EDM term}}$$

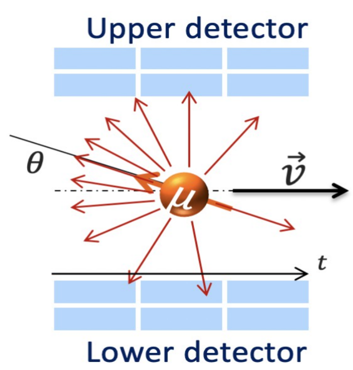
- Suppress g-2 term by setting $a\vec{B} = \left(a - \frac{1}{\gamma^2 - 1}\right)\frac{\vec{\beta}\times\vec{E}}{c}$
- Radial E-field $E_{\mathrm{f}} \approx aBc\beta\gamma^2$

$$\boxed{\vec{\omega}_e = \frac{\eta q}{2m}\left[\vec{\beta}\times\vec{B} + \frac{\vec{E}_{\mathrm{f}}}{c}\right]}$$

Precession frequency only due to EDM

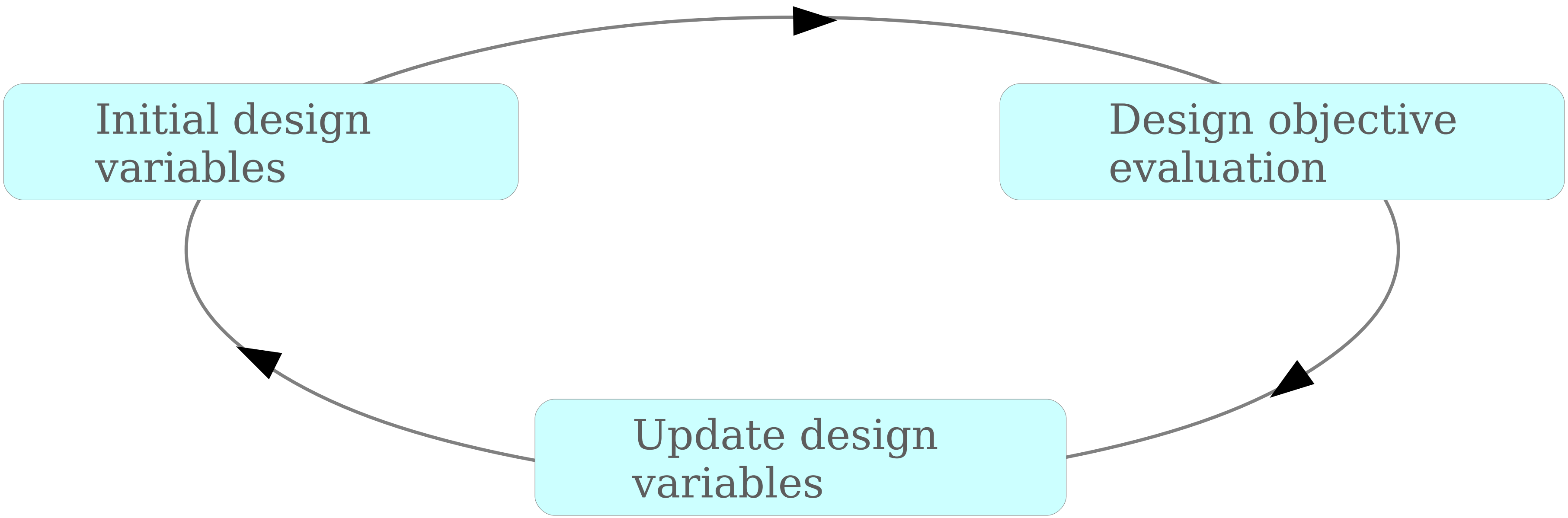
PSI muEDM Experiment





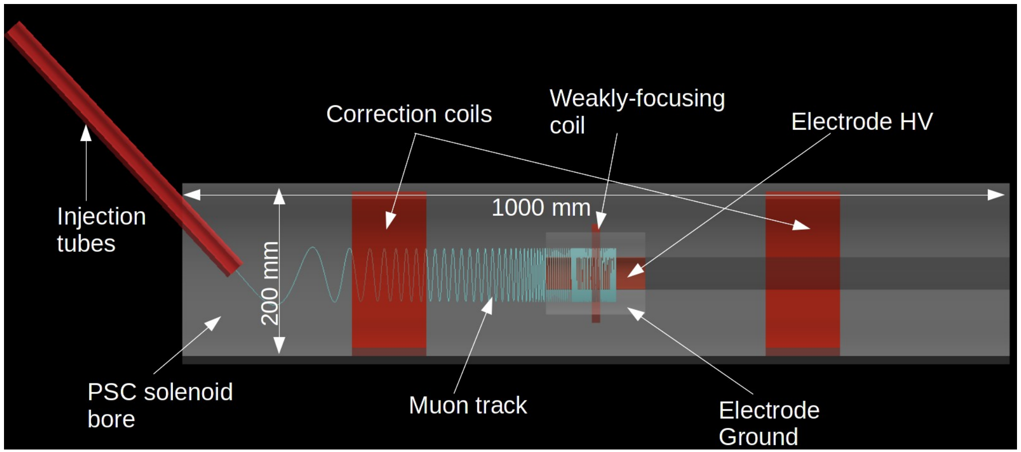
Asymmetry in number of detected positrons upstream vs downstream is proportional to EDM signal

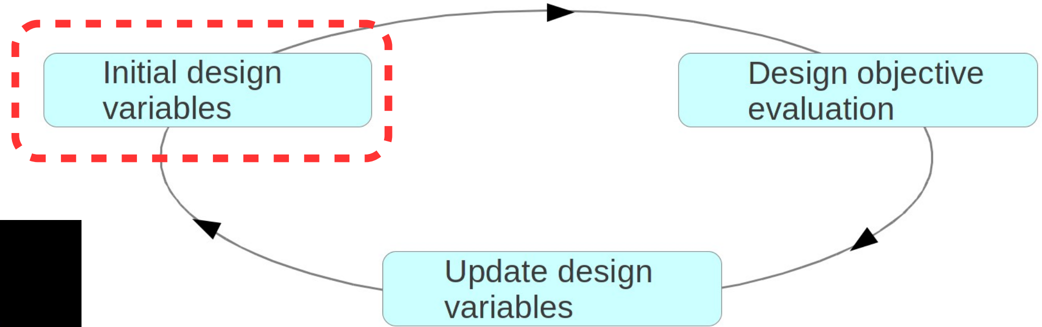
Design Optimization Layout

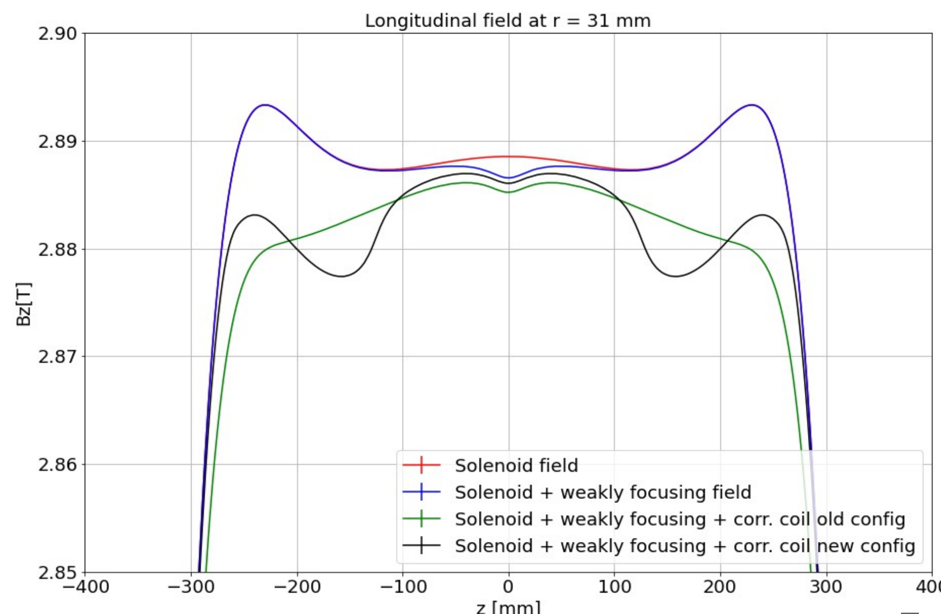


Design Optimization Layout

Design simulation in g4bl





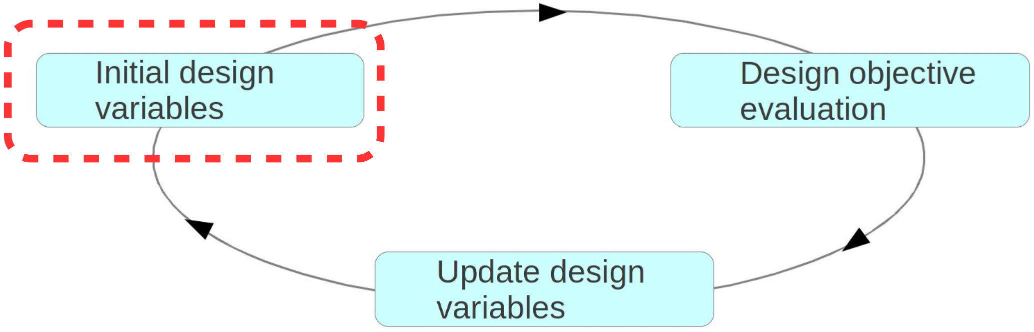


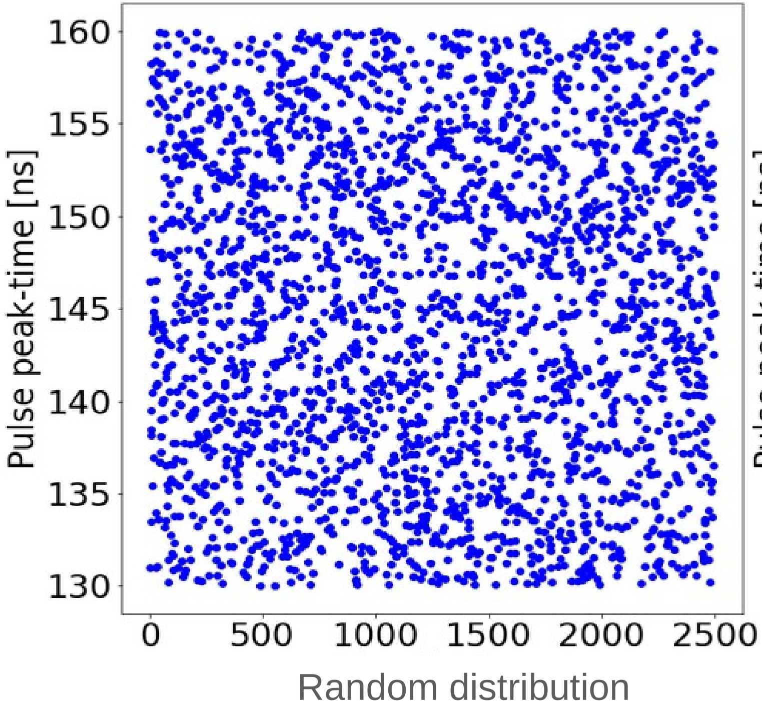
Design parameters:

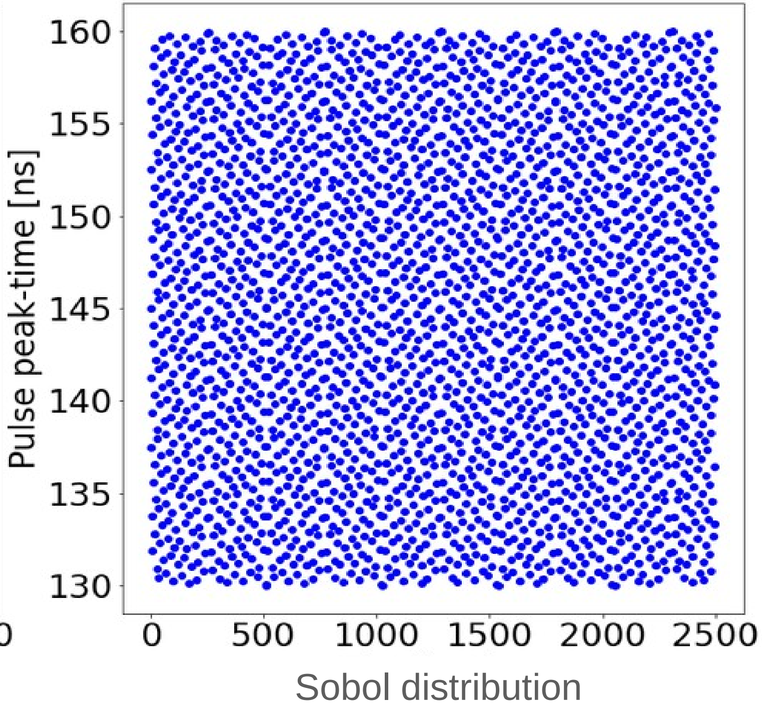
- Injection coordinates
- Magnetic field strength
- Correction coil features
- Weak-focusing coil features
- Kicker pulse features
- ……

Design Optimization Layout

- Sampling input variables
- Sobol distribution *(Sobol, 1967)*
- Maximum uniform spread

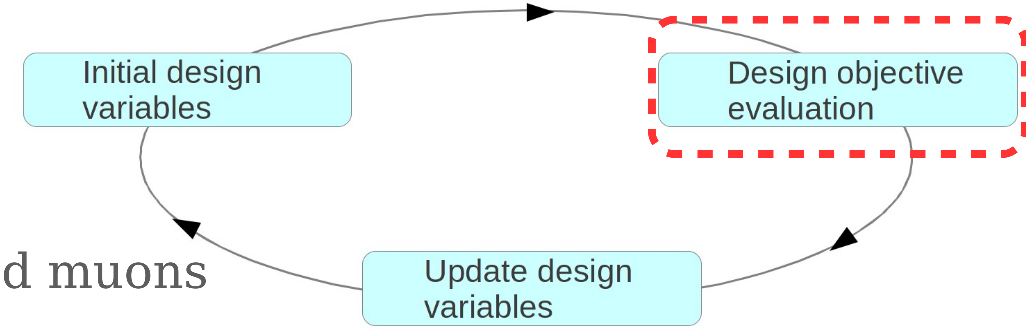


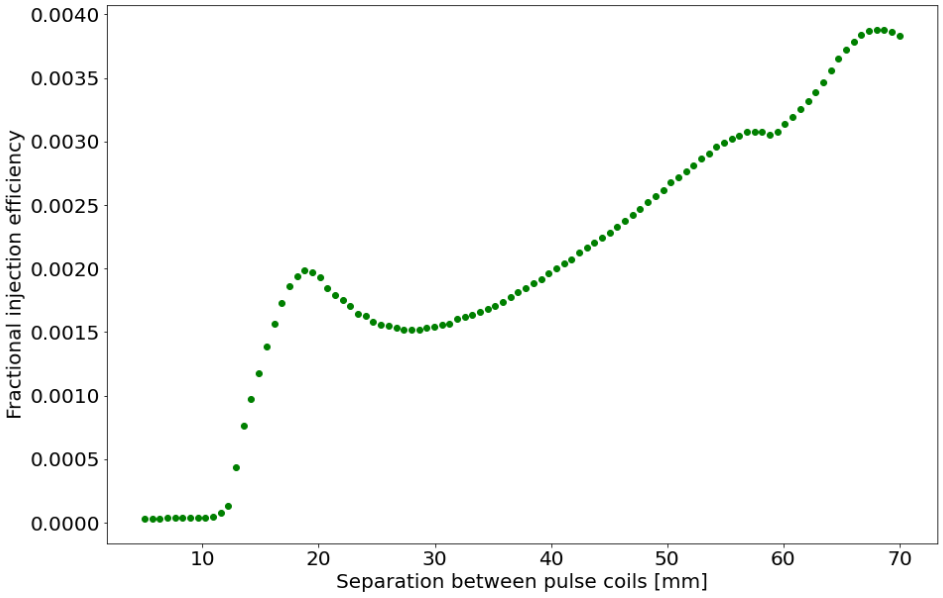


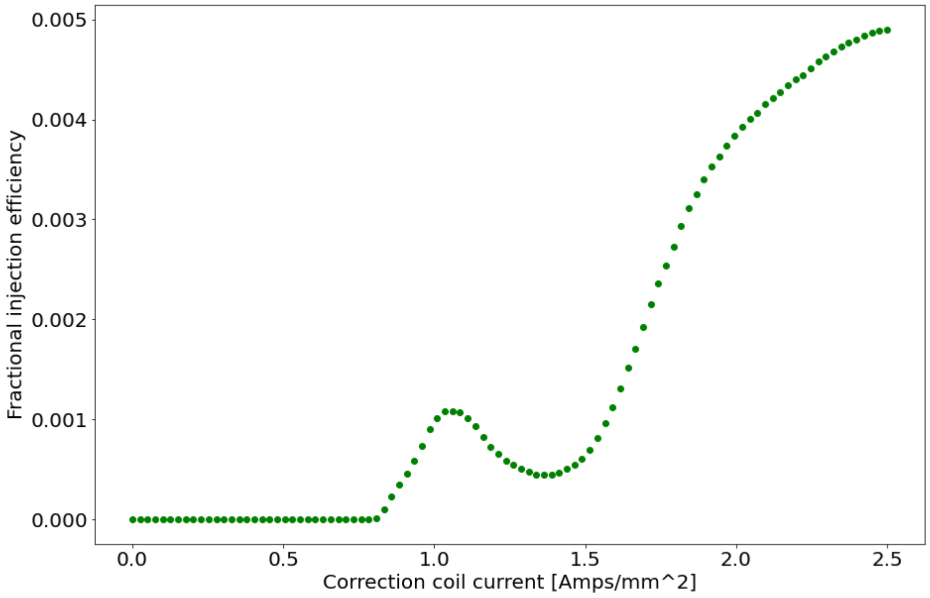


Design Optimization Layout

- Maximize injection efficiency
- Minimize power dissipation of setup
- Minimize polarization spread in stored muons
- ....

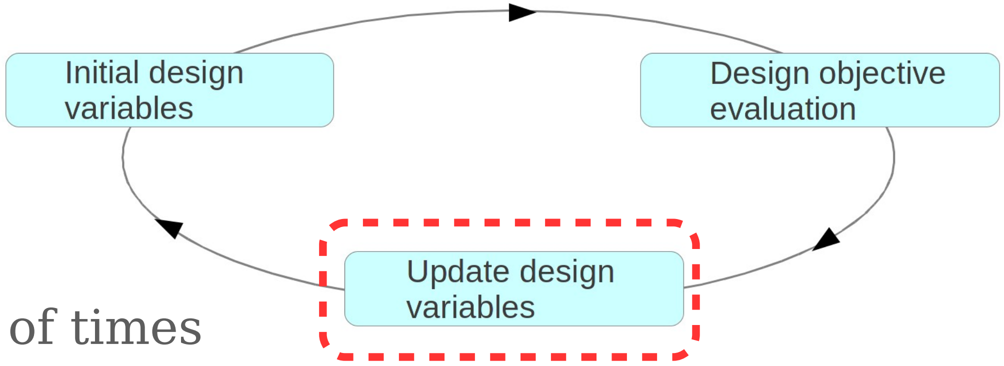




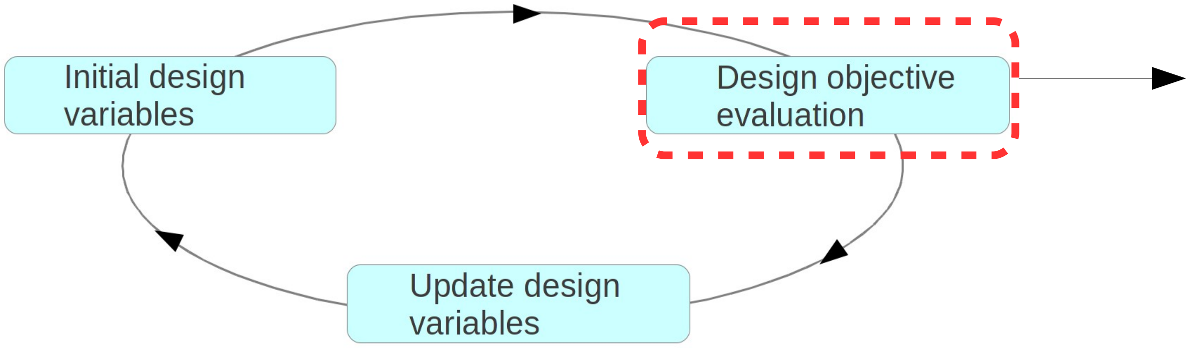


Design Optimization Layout

- Update design variables based on objective evaluation



- Repeat until optimal solution found
- Required to run simulation thousands of times
  → computationally expensive
- Replace physics simulation with approximation
  → surrogate model



Surrogate model for objective evaluation
→ Many ways
→ PCE and NN models explored

PCE Surrogate Model

- Polynomial Chaos Expansion (PCE) :

$$Y = \sum_{i=0}^{\infty} \alpha_i \Psi_i\left(\vec{x}\right)$$

$\boldsymbol{Y}$ → Model response (injection efficiency), $\boldsymbol{\Psi_i}$ → Orthogonal polynomials

x → input variables, $\boldsymbol{\alpha_i}$ → expansion coefficients

- Polynomial basis based on input variable distribution

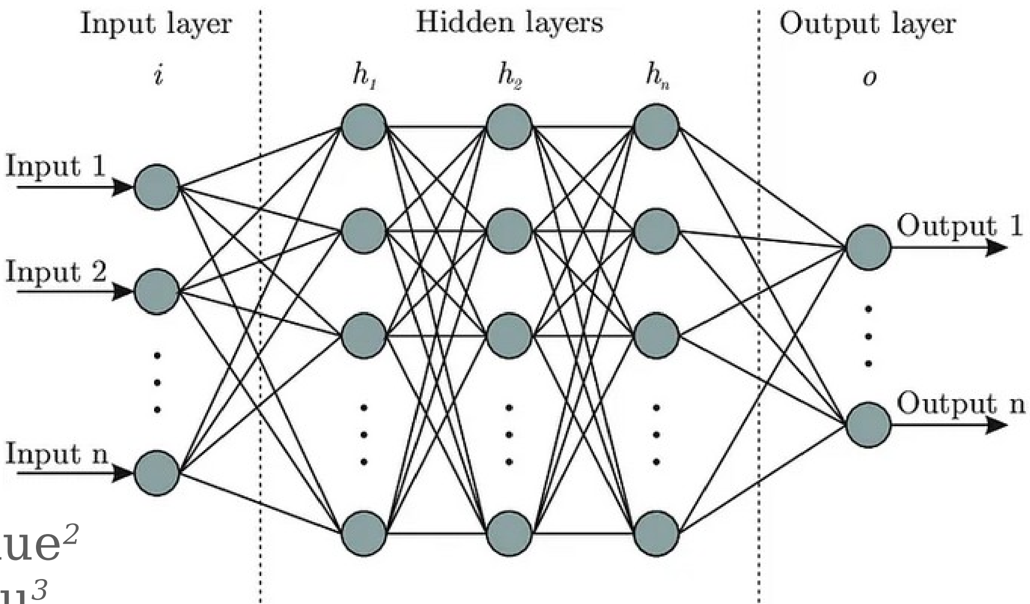
- Coefficients determined using regression based methods

$$\vec{\alpha} = \text{Argmin} \frac{1}{N} \sum_{j=1}^{N} \left\{ f\left(\vec{\xi}^{j}\right) - \sum_{i=0}^{P-1} \alpha_i \Psi_i\left(\vec{x}^{j}\right) \right\}^2$$

NN Surrogate Model

- Use the input (design) and output (objective) to train a neural network

- Hyper parameters:
  → no. of hidden layers = 8
  → no. of neurons/layer = 500
  → learning rate = 0.001
  → optimizer: Adam[1]
  → scheduler: ReduceLRonPlateaue[2]
  → activation function: LeakyReLu[3]

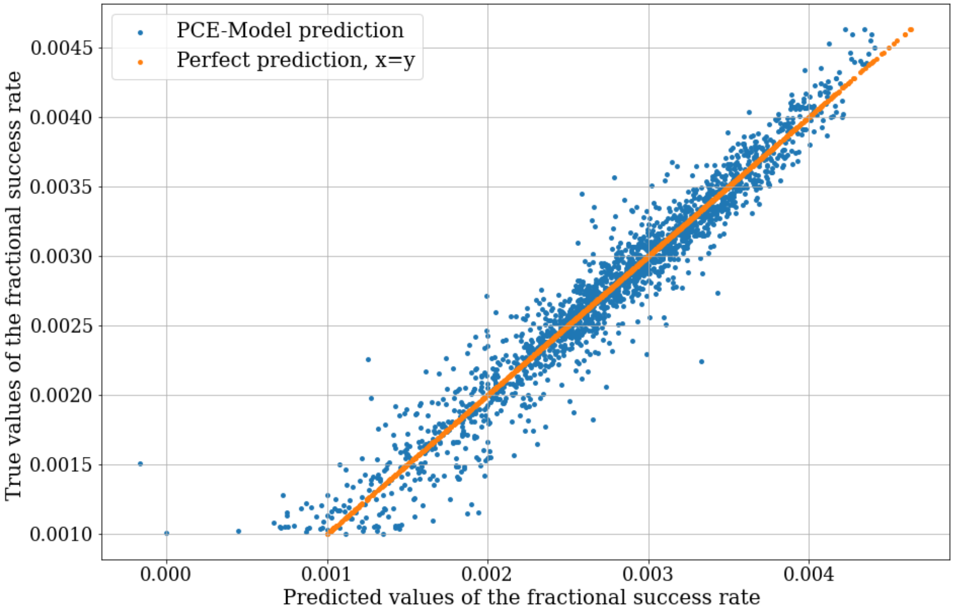


[1] *Kingma and Ba, 2014* [2] *Maas, 2013* [3] *K Developers, 2019*

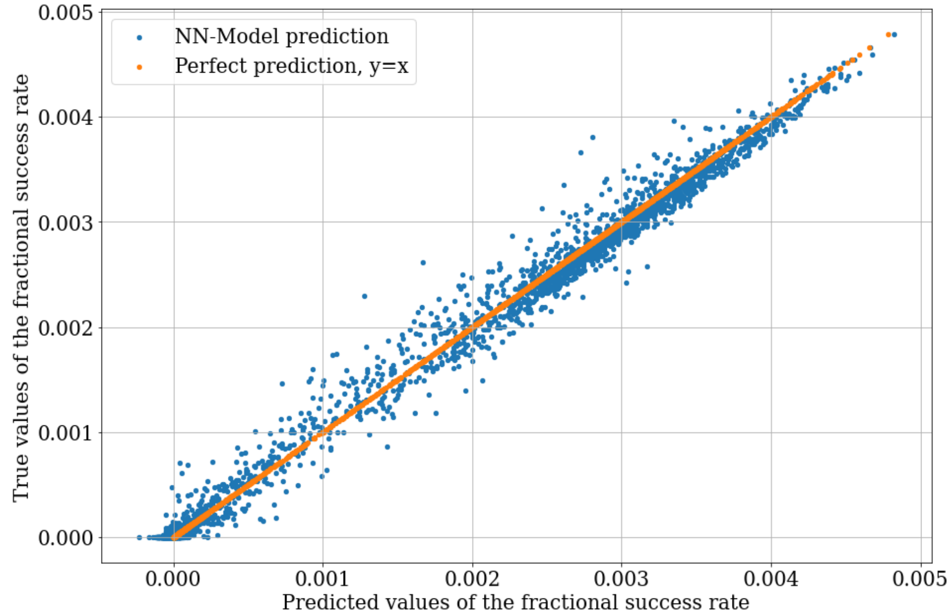
Surrogate Model Performance

Model performance for a 6 dimensional input space

(Kicker timing, Kicker strength, Corr coil position, Corr coil length, Corr coil thickness and Corr coil radius)

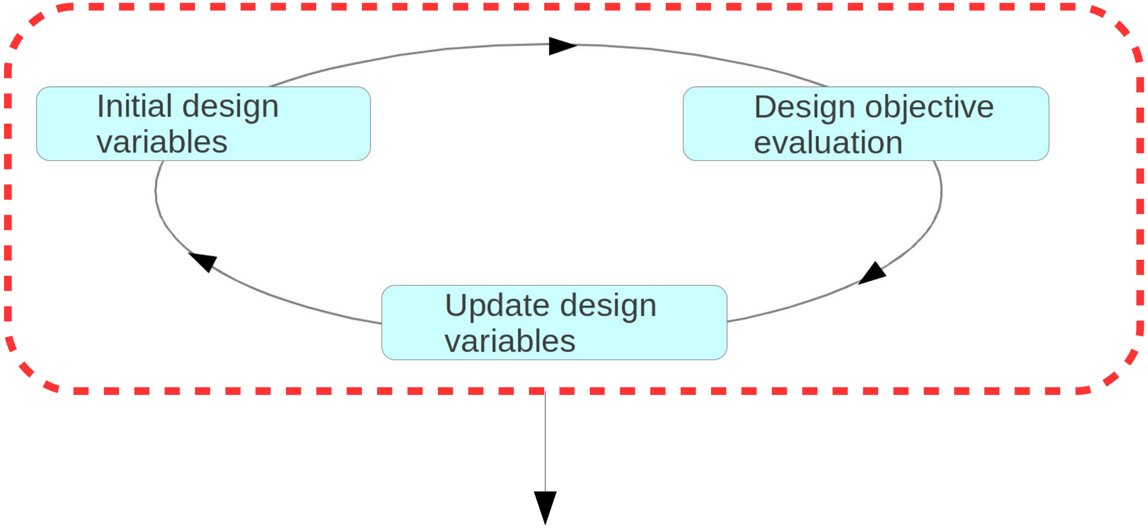


PCE Mean Square Error: 3.47 e-08



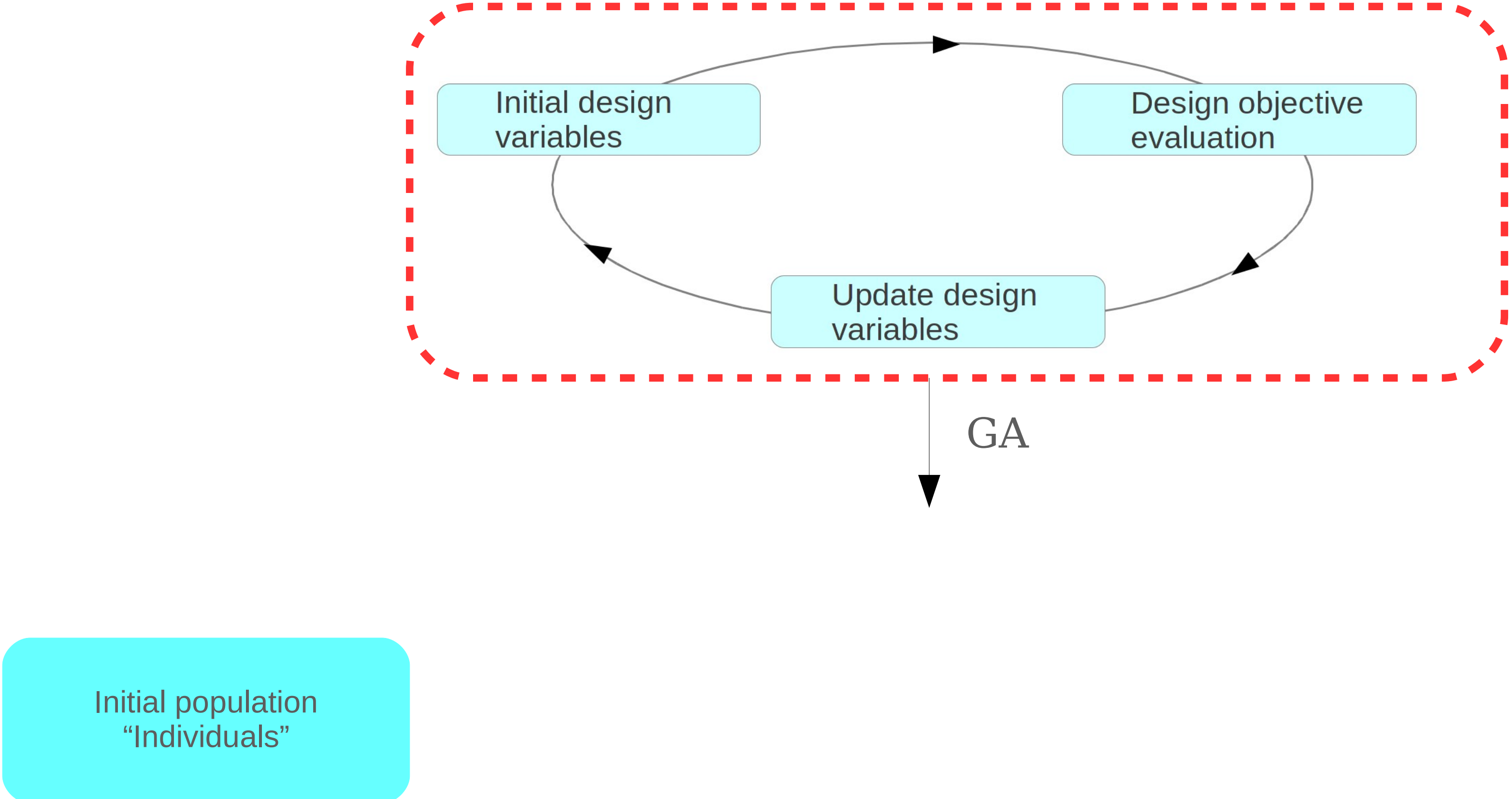
NN Mean Square Error: 1.88 e-08

Multi-objective Optimization

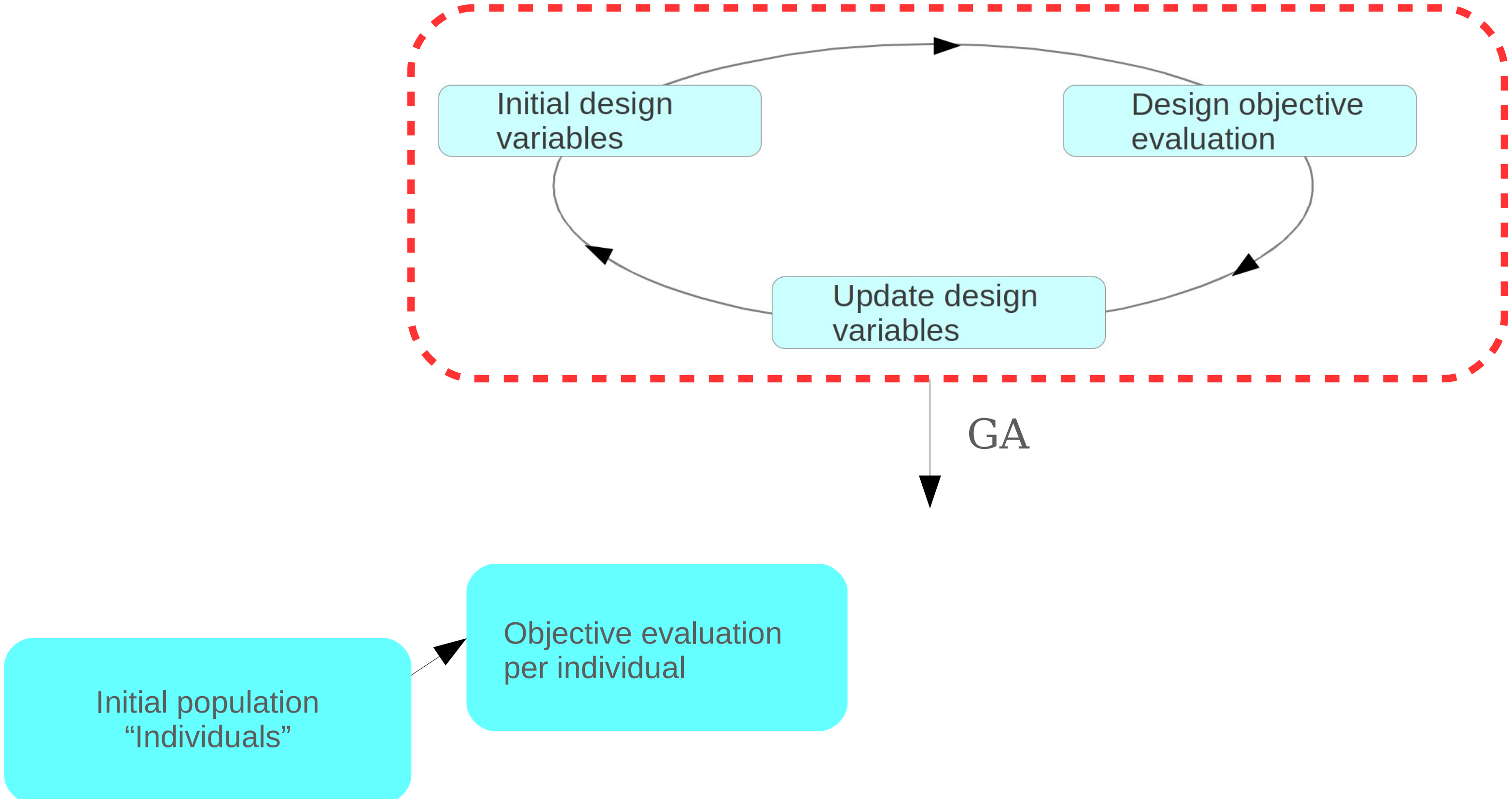


Genetic Algorithms (GA)

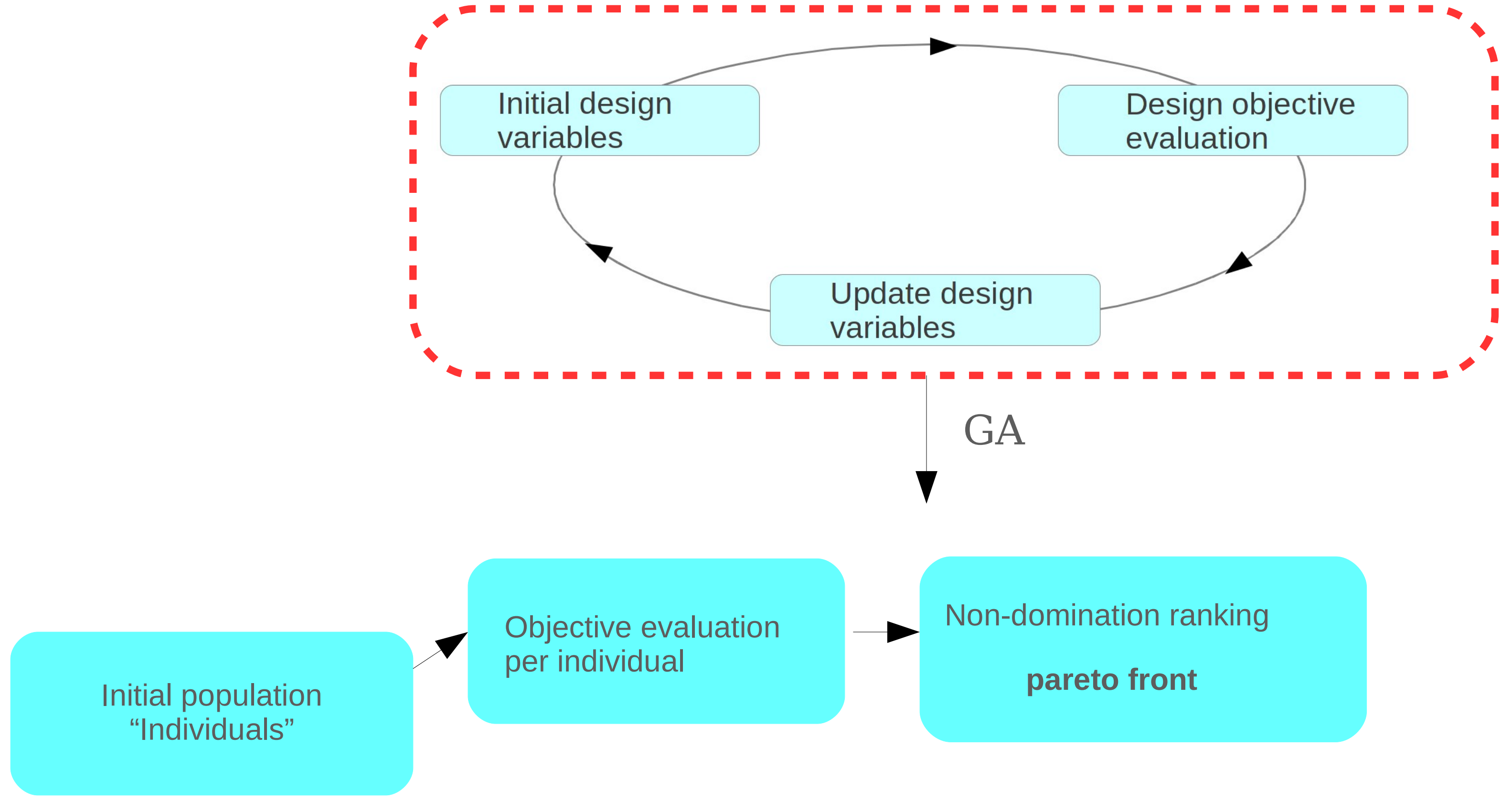
Genetic Algorithm



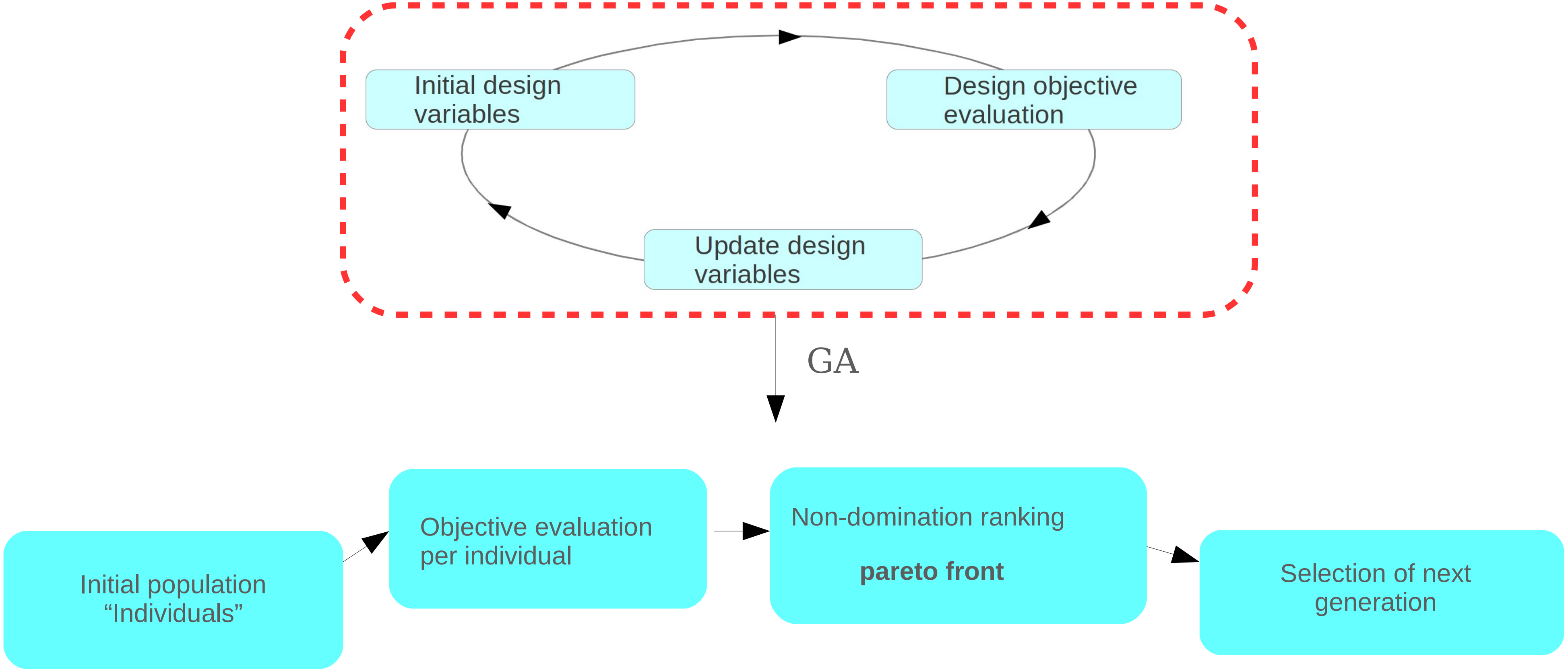
Genetic Algorithm



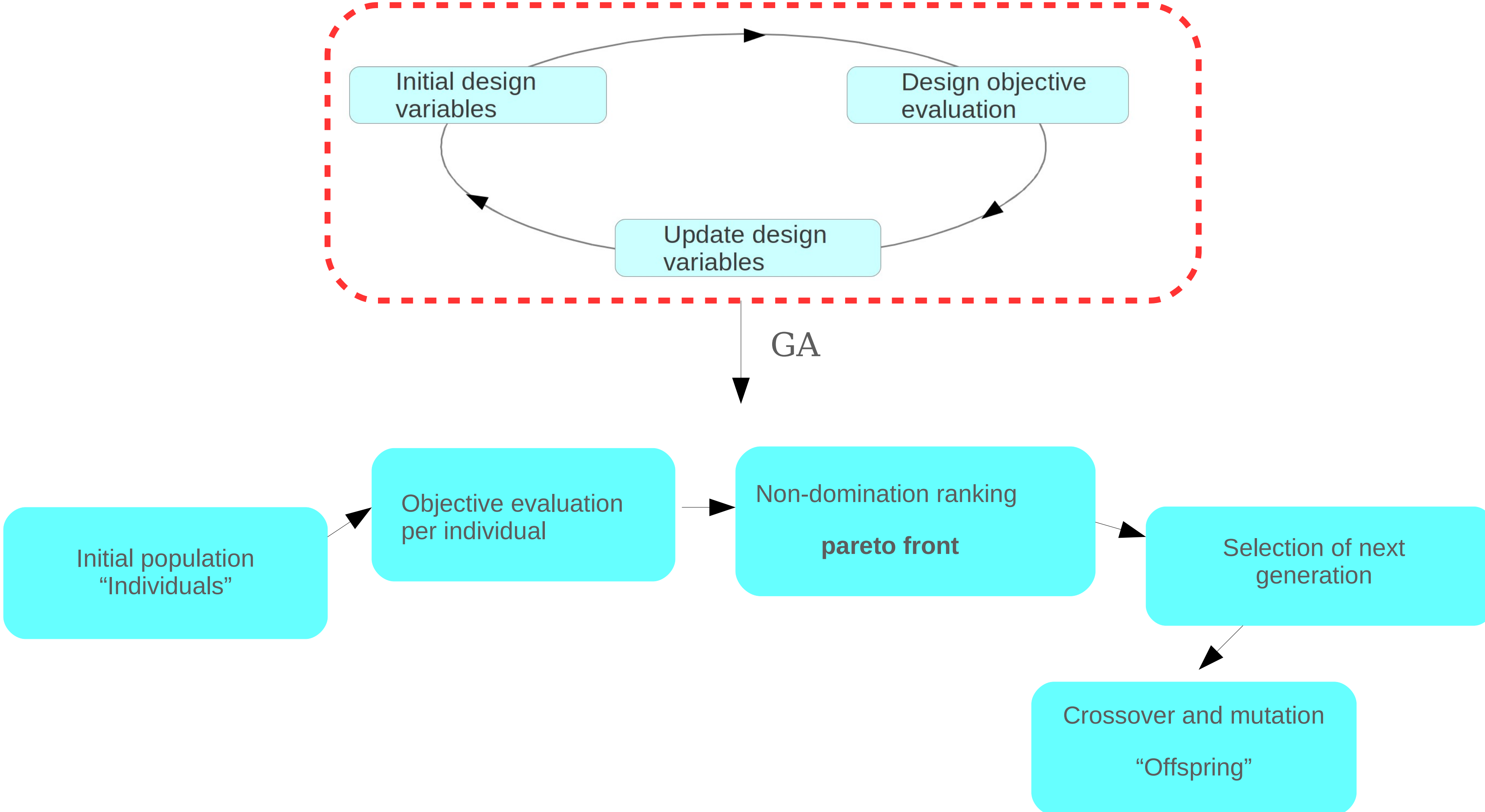
Genetic Algorithm



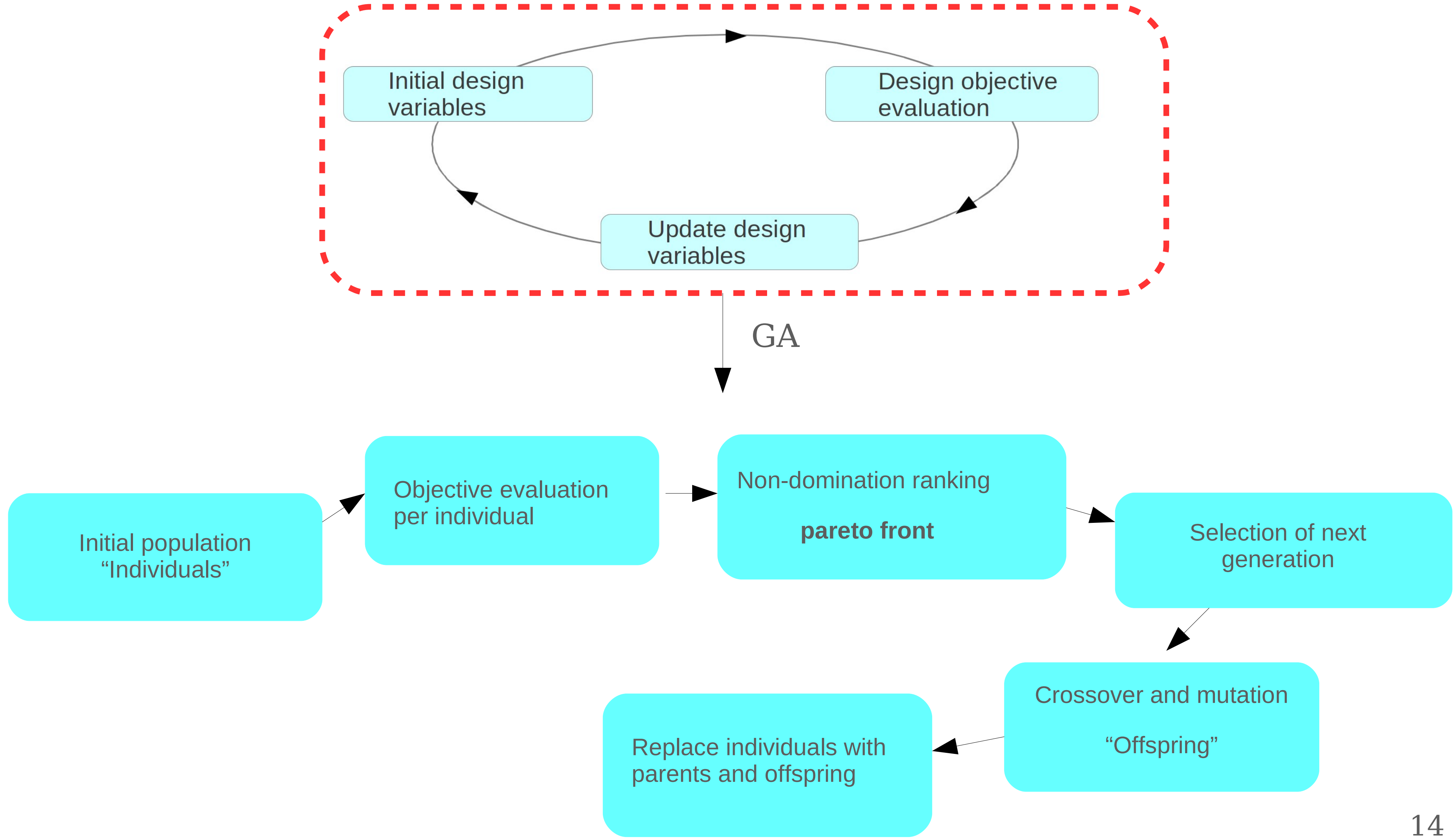
Genetic Algorithm



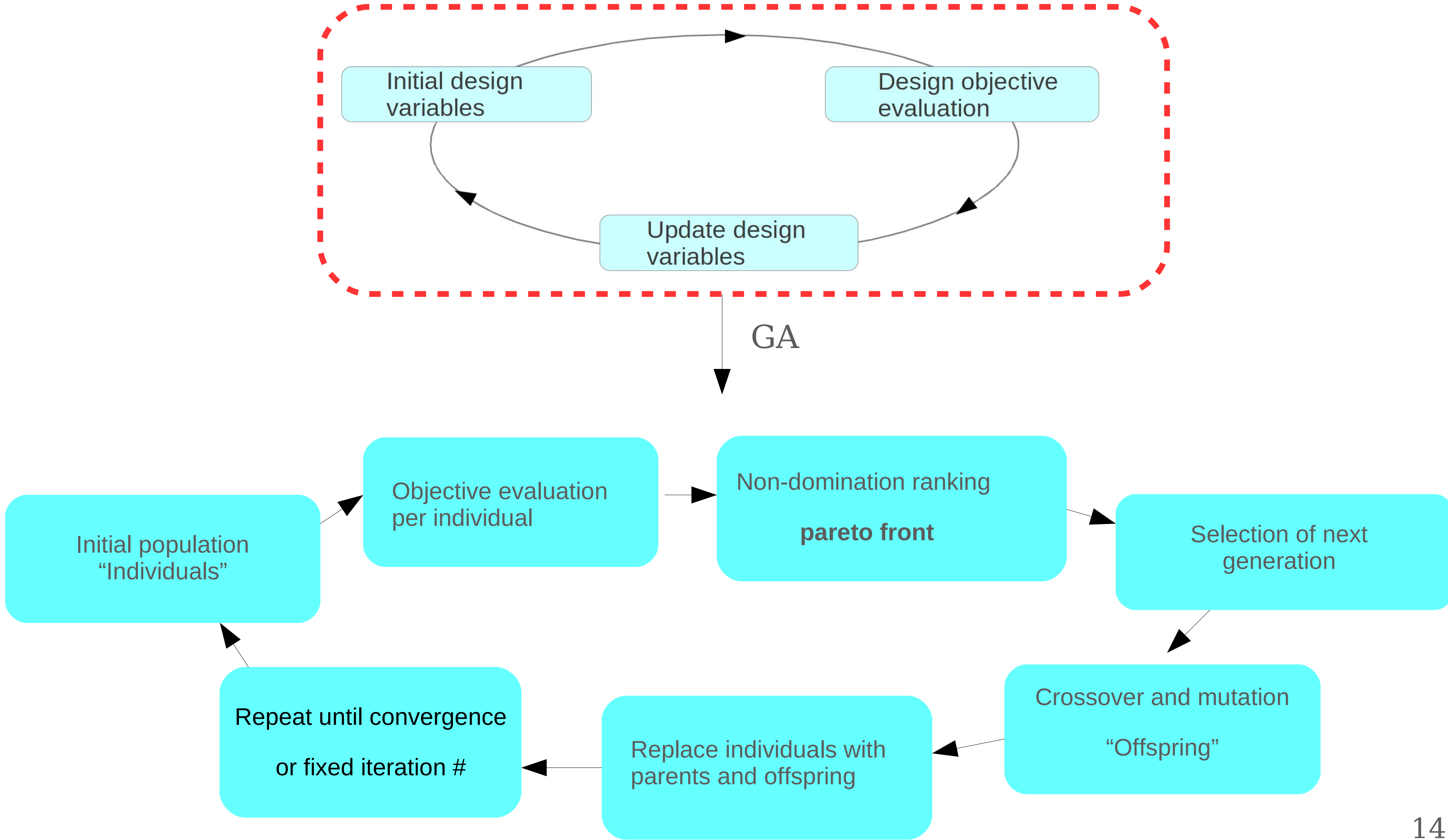
Genetic Algorithm



Genetic Algorithm

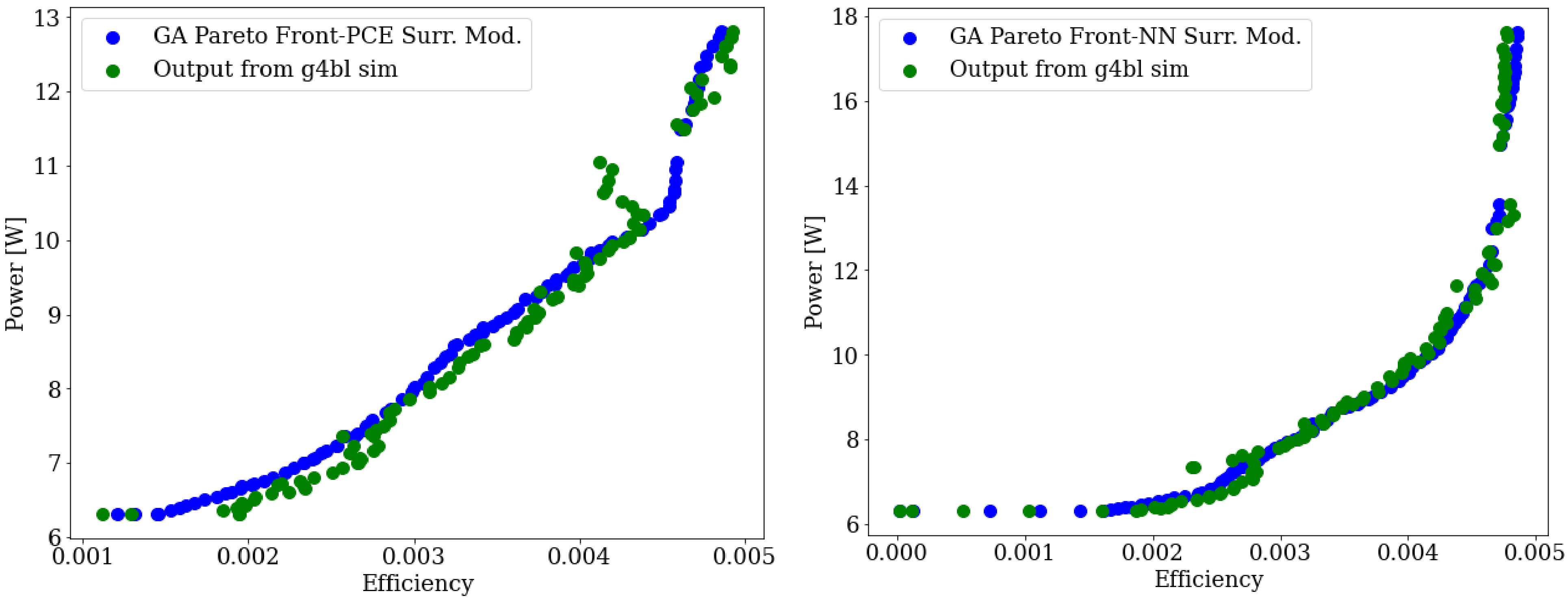


Genetic Algorithm



Surrogate model based NSGA-II[1] performance

Optimization to maximize Injection Efficiency/minimize Power Dissipation



- 10^3 speed up for PCE Surr and 10^4 speed up for NN Surr
- Agreement within 5% vs 2% for PCE/NN based GA performance for average injection efficiency of 0.35%

[1] *Deb 2002*

Summary

- PSI muEDM experiment will be most precise muon EDM measurement to date → setup needs to be carefully optimized

- Running simulations iteratively is bottleneck in optimization process

- Orders of magnitude speed up can be achieved by replacing physics simulation by surrogate model

- Genetic algorithm NSGA-II used to run multi-objective optimization

- PCE and NN surrogate models based GA investigated; $\sim 10^3$ speed up for PCE, $\sim 10^4$ for NN

- Plan to expand into Bayesian optimization where higher dimensional input space can be implemented with straightforward uncertainty quantification techniques

Acknowledgments

The muEDM Collaboration
(Spring meeting 2024)

- Computational resources: PSI Local High Performance Computing cluster, Merlin6, Siyuan-1 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University and the Euler cluster operated by the High Performance Computing group at ETH Zürich.

- Accelerator Modeling and Advanced Simulations (AMAS) group at PSI: A. Adelmann, S. Heinekamp and P. Juknevicius

- NN surrogate starting point: A. Holmberg Bachelor's Thesis ETH Zurich 2021

Project funded by

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs, Education and Research EAER
State Secretariat for Education, Research and Innovation SERI

Extra

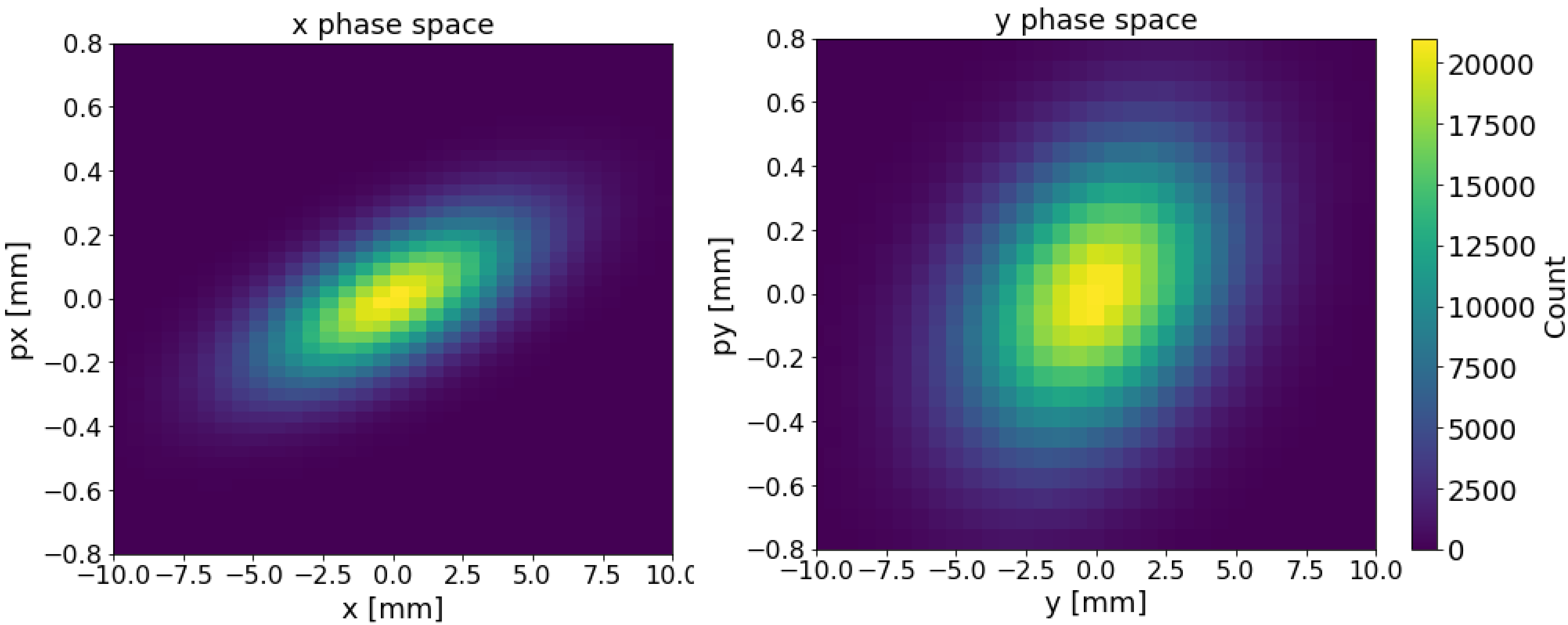
Polynomial Chaos Orthogonal Basis

The correspondence of the types of Wiener–Askey polynomial chaos and their underlying random variables ($N \geq 0$ is a finite integer).

	Random variables ζ	Wiener–Askey chaos $\{\Phi(\zeta)\}$	Support
Continuous	Gaussian	Hermite-chaos	$(-\infty, \infty)$
	gamma	Laguerre-chaos	$[0, \infty)$
	beta	Jacobi-chaos	$[a, b]$
	uniform	Legendre-chaos	$[a, b]$
Discrete	Poisson	Charlier-chaos	$\{0, 1, 2, \dots\}$
	binomial	Krawtchouk-chaos	$\{0, 1, \dots, N\}$
	negative binomial	Meixner-chaos	$\{0, 1, 2, \dots\}$
	hypergeometric	Hahn-chaos	$\{0, 1, \dots, N\}$

(Xiu and Karniadakis, 2002)

Total phase space after collimation

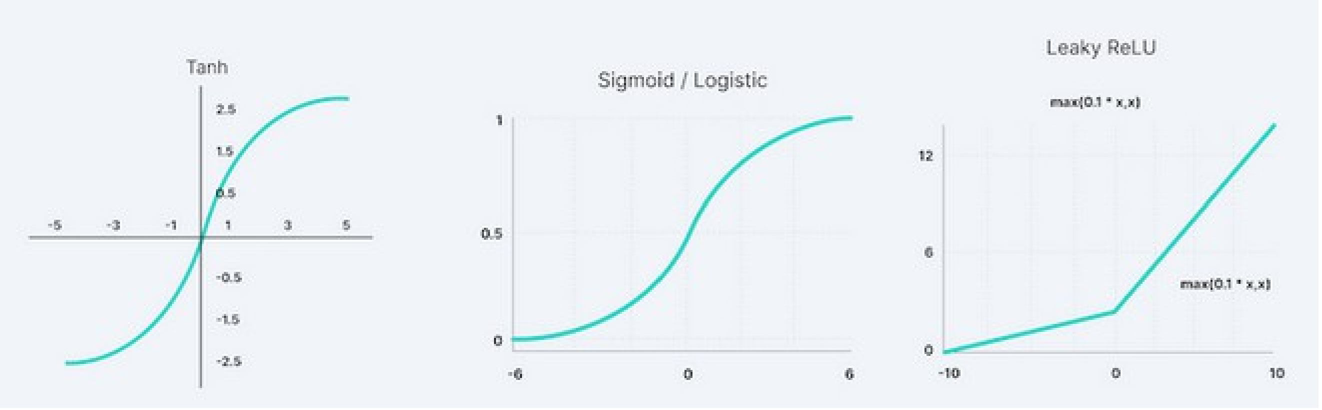


Neural Net hyperparameters

```
def __init__(self, input_dimension, output_dimension, n_hidden_layers,
             neurons, regularization_param, regularization_exp):
    super(net, self).__init__()
    # Number of input dimensions n
    self.input_dimension = input_dimension
    # Number of output dimensions m
    self.output_dimension = output_dimension
    # Number of neurons per layer
    self.neurons = neurons
    # Number of hidden layers
    self.n_hidden_layers = n_hidden_layers
    # Activation function
    self.activation = nn.LeakyReLU()
    #
    self.regularization_param = regularization_param
    #
    self.regularization_exp = regularization_exp

    self.input_layer = nn.Linear(self.input_dimension, self.neurons)
    self.hidden_layers = nn.ModuleList([nn.Linear(self.neurons, self.neurons) for _ in range(n_hidden_layers)])
    self.output_layer = nn.Linear(self.neurons, self.output_dimension)

    self.dropout = nn.Dropout(0.1)
```

```
# Model definition
my_network = net(input_dimension=x_train_norm.shape[1], output_dimension=y_train_norm.shape[1],
                 n_hidden_layers=8, neurons=512, regularization_param=0,
                 regularization_exp=0) #2  1e-5


# Random Seed for weight initialization
retrain = 134
# Xavier weight initialization
init_xavier(my_network, retrain)


optimizer_ = optim.Adam(my_network.parameters(), lr=1e-3)#, weight_decay=1e-5)
#optimizer_ = optim.LBFGS(my_network.parameters(), lr=0.1, max_iter=1,
#                         max_eval=50000, tolerance_change=1.0 * np.finfo(float).eps)

scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer_, mode='min', factor=0.5, patience=500000)
#scheduler = optim.lr_scheduler.StepLR(optimizer=optimizer_, step_size=50, gamma=0.5)

n_epochs = 200
```

Neural Net activation function



6-d optimization parameter bounds

```
bounds = {"T_Offset": [80, 98],
          "BPI": [0.35,0.80],
          "CC_Len": [88, 150],
          "CC_Ir": [40, 84],
          "CC_Thick":[7,15],
          "CC_Pos":[166,241]}
```