Methodology: Enhanced Genetic Algorithm with Adaptive Controllers (GAWorker)

1 Optimization Problem

Let $\boldsymbol{x} = (x_1, \ldots, x_d)^\top$ denote the decision vector of DVA parameters. For each gene i we have lower/upper bounds ℓ_i, u_i and an optional fixed value c_i. The feasible set is

$$x_i = c_i \ \ (i \in \mathcal{F}), \quad x_i \in [\ell_i, u_i] \ \ (i \notin \mathcal{F}),$$

where $\mathcal{F}$ is the index set of fixed parameters.

The FRF model maps parameters to a scalar performance indicator ("singular response") $s(\boldsymbol{x}; \Theta)$ over a frequency grid $\Omega = [\omega_{\min}, \omega_{\max}]$ with N_ω points, using main system parameters Θ. In addition, target criteria across masses yield percentage differences $\Delta_{m,k}(\boldsymbol{x})$ (absolute percent errors) for mass index $m \in \{1, \ldots, 5\}$ and criterion k.

The scalar fitness minimized by GAWorker is

$$f(\boldsymbol{x}) = \underbrace{|s(\boldsymbol{x}; \Theta) - 1|}_{\text{primary objective}} + \underbrace{\alpha \sum_{i=1}^{d} |x_i|}_{\text{sparsity penalty}} + \underbrace{\frac{1}{S} \sum_{m} \sum_{k} |\Delta_{m,k}(\boldsymbol{x})|}_{\text{percentage error term}}, \tag{1}$$

where $\alpha \geq 0$ is the sparsity weight (named `alpha` in code) and $S > 0$ is a scaling factor (`percentage_error_scale`). The algorithm terminates early when $\min f \leq \varepsilon$ (tolerance `ga_tol`) or when the generation budget is reached.

Diversity and statistics. Per generation, GAWorker computes

$$\mu = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} f_i, \qquad \sigma = \sqrt{\frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} f_i^2 - \mu^2}, \qquad cv = \frac{\sigma}{|\mu| + \varepsilon},$$

and a gene-level diversity

$$D = \frac{1}{d} \sum_{j=1}^{d} \min\Big(1, \max\big(0, \tfrac{\sigma_j}{u_j - \ell_j}\big)\Big),$$

where σ_j is the per-gene standard deviation across the population.

2 Initialization (Seeding)

GAWorker supports multiple seeding strategies with fixed-parameter handling:

- Random: $x_i \sim \mathcal{U}(\ell_i, u_i)$ for unfixed genes, $x_i = c_i$ if fixed.
- Sobol QMC: sample $\boldsymbol{z} \in [0,1]^d$ from a Sobol sequence and scale $\boldsymbol{x} = \boldsymbol{\ell} + \boldsymbol{z} \odot (\boldsymbol{u} - \boldsymbol{\ell})$; then overwrite fixed genes.
- Latin Hypercube (LHS): stratified sampling in $[0,1]^d$, then scale to $[\ell_i, u_i]$ and apply fixed genes.
- Memory seeding: propose from a replay buffer augmented with jitter and exploration.
- Best-of-pool: generate a large QMC pool, evaluate by (1), keep best with diversity stride.
- Neural seeding: propose $\boldsymbol{x}$ using a learned surrogate ensemble with acquisition UCB/EI and optional gradient refinement.

Why seeding matters (plain language). Before evolution begins, we must choose the first "generation" of designs to test. Think of seeding as laying different fishing nets across the ocean: better nets (seeding) catch better and more diverse fish (candidate solutions) sooner. Some nets spread widely and evenly (Sobol/LHS), some reuse good past spots (Memory), some try many and keep the best spread-out ones (Best-of-pool), and some use a learned map to suggest promising areas (Neural).

Fixed parameters. Sometimes a parameter must stay constant (e.g., a component you cannot change). We enforce that by setting both its lower and upper bound to the same value and keeping it fixed during creation, crossover, and mutation.

Seeding methods explained for newcomers

Random (uniform inside bounds). Like throwing darts blindfolded but making sure you always hit the board: every unfixed parameter is drawn uniformly within its allowed range. Pros: simple and unbiased. Cons: in higher dimensions it may leave big gaps (poor coverage).

Sobol (low-discrepancy) sampling. Sobol is a clever way to sprinkle points more evenly than random across a high-dimensional box. Imagine carefully placing dots so that every region gets fair attention. This improves coverage and often speeds up early progress compared to pure random draws.

Latin Hypercube Sampling (LHS). LHS divides each parameter range into equal segments and ensures you pick one value from each segment, combining them so that all parameters are well covered. Think of tasting a menu where you try one item from each category, rather than many from just one.

Memory seeding (replay + jitter + explore). We keep a lightweight "memory" of past good designs. Memory seeding mixes three ingredients: (i) replay top designs, (ii) add small random tweaks around them (local search), and (iii) add some fresh random points to keep exploring. This provides a strong starting population if you have historical runs.

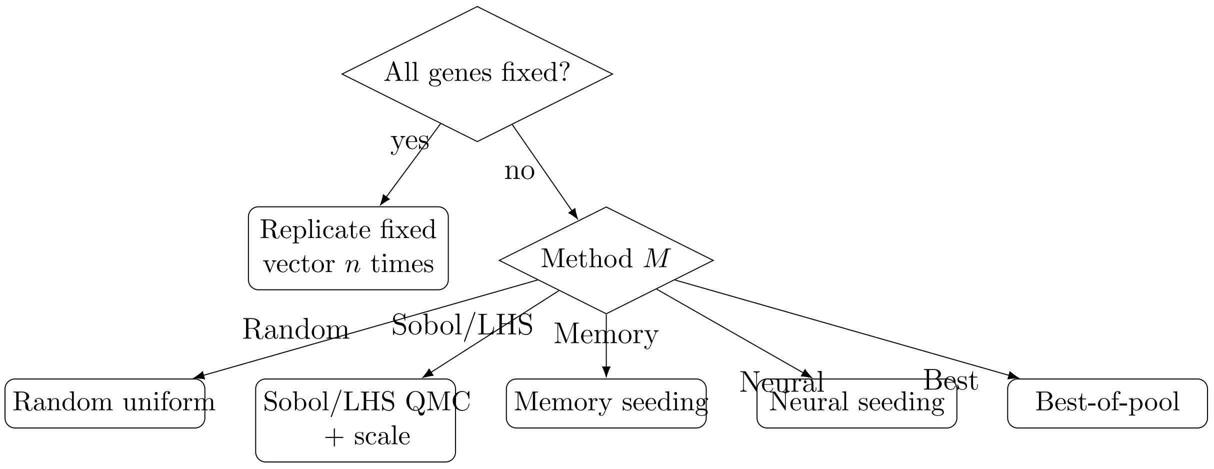
Best-of-pool (QMC pool + diversity stride). We generate a large, well-spread pool (e.g., with Sobol), score them quickly, sort from best to worst, and then pick with a "stride" so selected points are not bunched together. This combines exploitation (use the currently best) with diversity (keep them different) to avoid premature crowding.

Neural seeding (surrogate-guided). We train a fast approximation model (a small neural network ensemble) that learns from evaluated designs and predicts which new designs are promising. It balances two goals: (i) try designs that look best (exploitation) and (ii) try designs that reduce uncertainty or add diversity (exploration). This can jump-start the GA with smarter candidates.

Algorithm 1 GenerateSeedIndividuals(n)

```
Require: bounds $([\ell_i, u_i])_{i=1..d}$, fixed set $\mathcal{F}$ with values $c_i$, method $M$
 1: if $|\mathcal{F}| = d$ then
 2:     return $n$ copies of $(c_1, \dots, c_d)$
 3: end if
 4: if $M$ = RANDOM then
 5:     sample $n$ iid uniformly in bounds, apply fixed genes
 6:     return population
 7: end if
 8: if $M \in \{\text{SOBOL}, \text{LHS}\}$ then
 9:     initialize QMC engine for dimension $d$
10:     draw $n$ points $\boldsymbol{z} \in [0,1]^d$, scale to $[\ell_i, u_i]$, apply fixed
11:     return population
12: else if $M$ = MEMORY then
13:     query memory buffer for $n$ candidates, apply fixed, return
14: else if $M$ = BEST then
15:     draw a pool $\mathcal{U}$, evaluate $f(\cdot)$, sort ascending
16:     pick $n$ by diversity stride, return
17: else if $M$ = NEURAL then
18:     propose $n$ via surrogate acquisition (§8)
19:     return
20: else
21:     fallback to RANDOM
22: end if
```

Seeding decision tree.



2.1 Neural seeding in detail (beginner-friendly)

Intuition. Evaluating a design is relatively expensive (it runs the FRF and computes targets). A small neural network can learn a quick approximation from the designs we already evaluated. We then ask this learned model where to look next, prioritizing designs that are predicted to be good, while keeping some room to explore unfamiliar regions.

Model (ensemble surrogate). We use an ensemble of M small feed-forward networks. Each network takes normalized parameters $\tilde{\boldsymbol{x}} \in [0,1]^d$ and outputs a predicted fitness $\hat{f}_m(\boldsymbol{x})$. The ensemble provides

$$\mu(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} \hat{f}_m(\boldsymbol{x}), \qquad \sigma(\boldsymbol{x}) = \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(\hat{f}_m(\boldsymbol{x}) - \mu(\boldsymbol{x}))^2},$$

which we interpret as mean and uncertainty. Architecturally, each network has:

- Input layer of size d (one neuron per parameter),
- L hidden layers (e.g., ReLU) with width H and dropout for regularization,
- A single linear output predicting fitness.

Training (fast, incremental). Every generation we add the latest evaluated designs $\{(\boldsymbol{x}^{(j)}, f^{(j)})\}$ to a training buffer. We normalize inputs to $[0,1]$ using bounds. Each network trains a few short epochs (or up to a time cap) with mean squared error loss $\mathcal{L} = \frac{1}{B}\sum(\hat{f} - f)^2$ on small batches. This keeps training overhead low while adapting to new data.

Acquisition functions (choose where to sample). For minimization we support:

$$\mathrm{UCB}(\boldsymbol{x}) = \mu(\boldsymbol{x}) - \beta\,\sigma(\boldsymbol{x}) \qquad (\beta > 0:\ \text{trade-off mean vs.}$$

$$\mathrm{EI}(\boldsymbol{x}) = \mathbb{E}[\max(0, f^* - F(\boldsymbol{x}))] \approx (f^* - \mu)\,\Phi(z) + \sigma\,\phi(z),\ z = \frac{f^* - \mu}{\sigma},$$

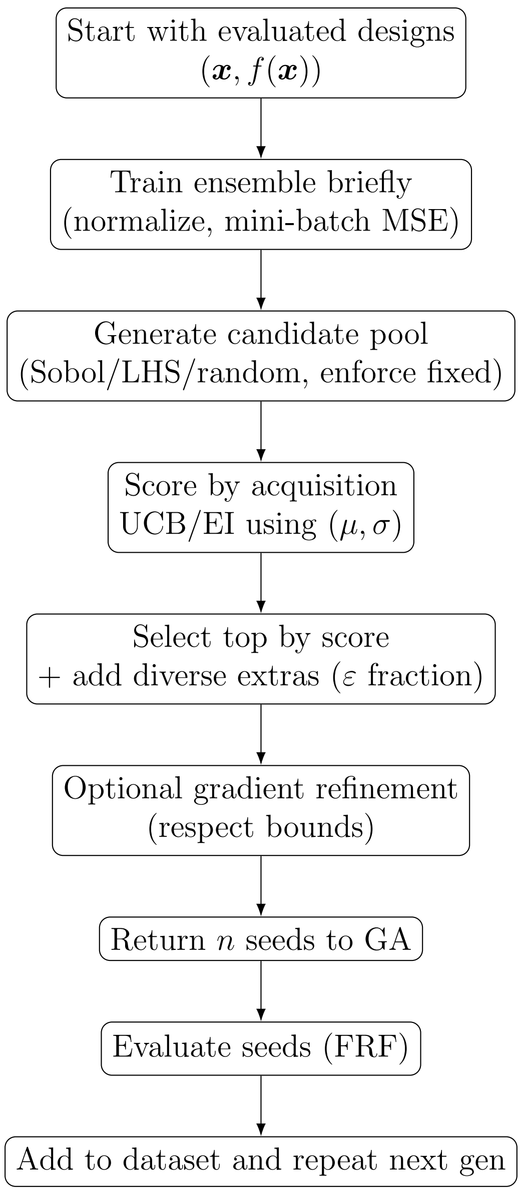
where f^* is the best fitness so far, and Φ, ϕ are the standard normal CDF and PDF. UCB is simple and robust; EI directly models expected improvement.

Candidate generation and selection. We generate a pool of size ρn (e.g., using Sobol), enforce fixed genes, and score candidates by the acquisition. We select $(1-\varepsilon)n$ best by acquisition and fill the remaining εn by a diversity heuristic (choose points far from those already selected). The exploration fraction ε can adapt with stagnation: if improvement stalls, we increase exploration to escape local traps.

Algorithm 2 NeuralSeeder: propose(n)

```
Require: pool multiplier $\rho > 1$, exploration fraction $\varepsilon \in [0,1]$, acquisition $\mathcal{A}(\cdot)$
1: build pool $\mathcal{U}$ of size $\lceil \rho n \rceil$ in $[\ell_i, u_i]$, apply fixed genes
2: score each $\boldsymbol{x} \in \mathcal{U}$ by $a(\boldsymbol{x}) \leftarrow \mathcal{A}(\boldsymbol{x})$ (lower is better)
3: $k \leftarrow \lfloor (1-\varepsilon)n \rfloor$; pick $k$ lowest $a(\boldsymbol{x})$ to form $\mathcal{S}$
4: while $|\mathcal{S}| < n$: add the point in $\mathcal{U} \setminus \mathcal{S}$ that maximizes distance to $\mathcal{S}$ (diversity)
5: optional: gradient refinement of $\mathcal{S}$ within bounds
6: return $\mathcal{S}$
```

Neural seeding workflow.



Start with evaluated designs $(\boldsymbol{x}, f(\boldsymbol{x}))$
Train ensemble briefly (normalize, mini-batch MSE)
Generate candidate pool (Sobol/LHS/random, enforce fixed)
Score by acquisition UCB/EI using (μ, σ)
Select top by score + add diverse extras (ε fraction)
Optional gradient refinement (respect bounds)
Return n seeds to GA
Evaluate seeds (FRF)
Add to dataset and repeat next gen

When to use which seeding method?

- Use **Random** for quick tests or low dimensions.
- Use **Sobol/LHS** for broad, even coverage when you have no prior data.
- Use **Memory** to leverage previous runs or restart from good regions.
- Use **Best-of-pool** when you can afford a larger initial screening and want both quality and spread.
- Use **Neural** when evaluations are costly and you want the GA to start with informed, high-quality candidates that balance exploration and exploitation.

3 Evolutionary Operators

Selection. Tournament selection of size 3.

Crossover (Blend). For parents $\boldsymbol{x}, \boldsymbol{y}$ and $\beta \in [0,1]$ (code uses $\beta = 0.5$), offspring gene-wise

$$x'_i = \text{clip}\big(\beta x_i + (1-\beta) y_i,\ [\ell_i, u_i]\big), \quad x'_i \leftarrow c_i\ (i \in \mathcal{F}).$$

Mutation. With probability p_{mut} per individual, and per-gene probability p_{ind} (`indpb`), apply

$$x_i \leftarrow \text{clip}\big(x_i + \delta_i,\ [\ell_i, u_i]\big), \qquad \delta_i \sim \mathcal{U}\big(-\eta\Delta_i,\ \eta\Delta_i\big),$$

where $\Delta_i = u_i - \ell_i$ and η is the dynamic mutation scale (`mutation_scale`). Fixed genes are skipped.

Algorithm 3 One Generation (selection, crossover, mutation, repair)

```
1:  $\mathcal{O} \leftarrow$ clone(SelectTournament($\mathcal{P}, |\mathcal{P}|, t = 3$))
2:  for all pairs $(o_1, o_2) \in \mathcal{O}$ do
3:      if Rand() $< p_{\text{cx}}$ then
4:          $(o_1, o_2) \leftarrow$ Blend($o_1, o_2, \beta$); repair to bounds, apply fixed
5:      end if
6:  end for
7:  for all $o \in \mathcal{O}$ do
8:      if Rand() $< p_{\text{mut}}$ then
9:          $o \leftarrow$ Mutate($o, \eta$); repair to bounds, apply fixed
10:     end if
11: end for
12: evaluate invalid $o \in \mathcal{O}$ using Algorithm 4
13: replace $\mathcal{P} \leftarrow \mathcal{O}$
```

4 Fitness Evaluation

Algorithm 4 EvaluateIndividual($\boldsymbol{x}$)

```
1: if paused/aborted then
2:     return large penalty
3: end if
4: $s \leftarrow \mathrm{FRF}(\boldsymbol{x}; \Theta, \Omega)$; if missing, sum composite measures
5: $f_1 \leftarrow |s-1|$; $f_2 \leftarrow \alpha \sum_i |x_i|$
6: $E \leftarrow \sum_m \sum_k |\Delta_{m,k}(\boldsymbol{x})|$
7: return $f(\boldsymbol{x}) = f_1 + f_2 + E/S$
```

5 Adaptive Hyperparameter Control

Let $(p_{\mathrm{cx}}, p_{\mathrm{mut}}, N)$ denote crossover prob., mutation prob., and population size.

5.1 Legacy heuristic (success/diversity driven)

Maintain an EMA of success rate $\hat{s}$ and gene diversity $\hat{D}$. Every heartbeat or upon stagnation:

$$\begin{aligned}
&\text{if } \hat{s} < 0.9s^* : \ p_{\mathrm{mut}} \leftarrow \min(\bar{p}_{\mathrm{mut}},\, 1.25\, p_{\mathrm{mut}}),\\
&\text{if } \hat{s} > 1.1s^* : \ p_{\mathrm{mut}} \leftarrow \max(\underline{p}_{\mathrm{mut}},\, 0.8\, p_{\mathrm{mut}}),\\
&\text{if } D \ll D^* : \ p_{\mathrm{mut}} \uparrow,\ p_{\mathrm{cx}} \downarrow,\ \eta \uparrow; \quad \text{if } D \gg D^* : \ p_{\mathrm{cx}} \uparrow,\ p_{\mathrm{mut}} \downarrow,\ \eta \downarrow .
\end{aligned}$$

5.2 ML Bandit controller (UCB)

Define a discrete action set $\mathcal{A} = \{(\delta_{cx}, \delta_{mut}, \rho)\}$ to scale $(p_{\mathrm{cx}}, p_{\mathrm{mut}}, N)$. For action a, maintain count n_a and average reward $\bar{R}_a$. At time t select

$$a_t = \arg\max_{a \in \mathcal{A}} \ \underbrace{\left(w_h\, \bar{R}_a + w_c\, R_t\right)}_{\text{blended exploitation}} + c\sqrt{\tfrac{\ln t}{n_a}}.$$

The per-generation reward mirrors the code:

$$R_t = \frac{\max(0, f^*_{t-1} - f^*_t)}{\Delta t \cdot \max(1, \#\mathrm{evals})} - \lambda\, |cv_t - cv^*|.$$

Apply the selected action as

$$p_{\mathrm{cx}} \leftarrow \mathrm{clip}\big(p_{\mathrm{cx}}(1+\delta_{cx}), [\underline{p}_{\mathrm{cx}}, \bar{p}_{\mathrm{cx}}]\big),\ p_{\mathrm{mut}} \leftarrow \mathrm{clip}\big(p_{\mathrm{mut}}(1+\delta_{mut}), [\underline{p}_{\mathrm{mut}}, \bar{p}_{\mathrm{mut}}]\big),\ N \leftarrow \mathrm{round}\,\mathrm{clip}(\rho N, [N_{\mathrm{min}}, N$$

Algorithm 5 ML Bandit step

```
1: compute $(\mu, \sigma, cv)$ and best $f^*_t$; measure $\Delta t$, evals
2: for all $a \in \mathcal{A}$ do
3:     $U_a \leftarrow (w_h \bar{R}_a + w_c R_t) + c\sqrt{\ln t / n_a}$ (if $n_a = 0$ set $U_a = +\infty$)
4: end for
5: pick $a_t = \arg\max U_a$; update $(p_{\mathrm{cx}}, p_{\mathrm{mut}}, N)$; resize population if needed
6: after generation: observe $R_t$ and update $\bar{R}_{a_t}$, $n_{a_t}$
```

5.3 RL controller (Q-learning)

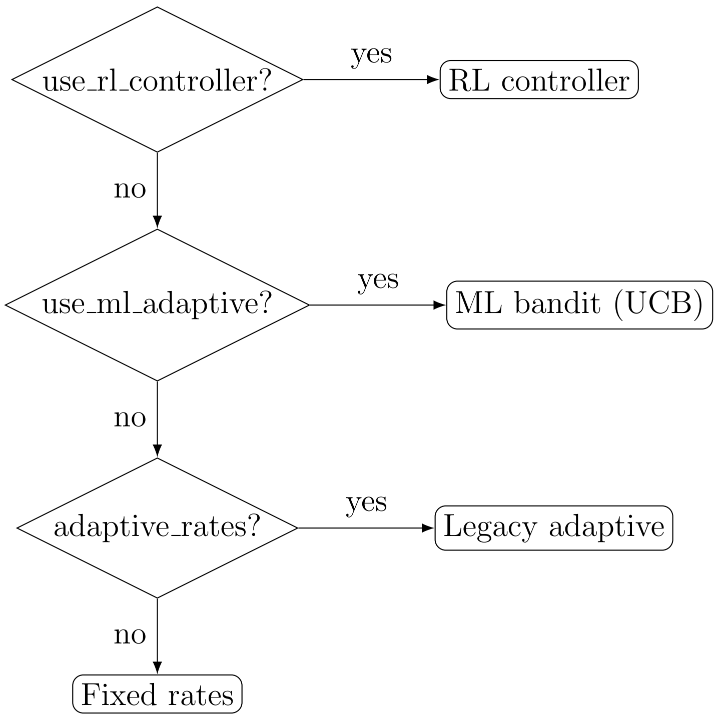
States $s \in \{0, 1\}$ encode coarse progress; actions are as above. Epsilon-greedy selection and tabular update

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right].$$

Algorithm 6 RL step

```
1: with prob. $\varepsilon$ pick random $a$, else $\arg\max_a Q(s, a)$
2: apply $a$, run generation, observe reward $r$ and next state $s'$
3: $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
4: $s \leftarrow s'$, $\varepsilon \leftarrow \varepsilon \cdot \text{decay}$
```

Controller decision tree.



6 Surrogate-Assisted Screening

When the genetic algorithm (GA) produces new offspring that have not yet been evaluated (i.e., "invalid" individuals), and there is a sufficiently large history of past evaluated solutions, the GAWorker uses a surrogate-assisted screening process to decide which candidates to actually evaluate with the expensive objective function. This process is designed to save computation by focusing evaluations on the most promising or diverse candidates.

Surrogate Model: GAWorker constructs a simple k-nearest neighbors (kNN) surrogate model using the archive of previously evaluated individuals. All parameter vectors $\boldsymbol{x}$ are first normalized to the unit hypercube $[0, 1]^d$ using

$$\tilde{x}_i = \frac{x_i - \ell_i}{u_i - \ell_i}$$

where ℓ_i and u_i are the lower and upper bounds for parameter i. For any candidate solution $\boldsymbol{z}$, its predicted fitness $\hat{f}(\boldsymbol{z})$ is computed as the average fitness of its k nearest neighbors in the normalized space:

$$\hat{f}(\boldsymbol{z}) = \frac{1}{k} \sum_{j \in \mathcal{N}_k(\tilde{\boldsymbol{z}})} y_j, \qquad y_j = f(\boldsymbol{x}^{(j)})$$

where $\mathcal{N}_k(\tilde{\boldsymbol{z}})$ denotes the indices of the k closest archived samples to $\tilde{\boldsymbol{z}}$.

Screening Procedure:

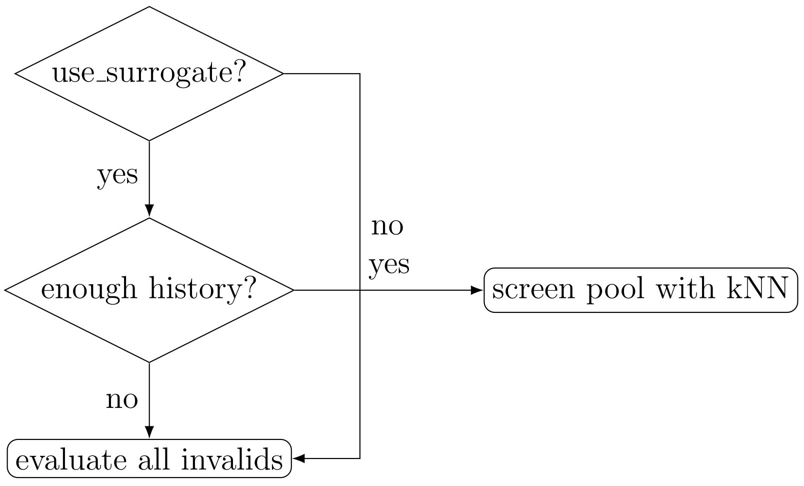
- *Pool Construction:* GAWorker generates a pool of candidate offspring by applying crossover, mutation, and cloning until the pool size reaches $\lceil \rho q \rceil$, where q is the number of evaluations to perform and $\rho > 1$ is a pool factor (e.g., 2).
- *Surrogate Prediction:* For each candidate in the pool, the kNN surrogate predicts its fitness $\hat{f}$.
- *Exploitation vs. Exploration:* The q candidates to be evaluated are chosen as follows:
  - A fraction $q_e = \lfloor (1-\xi) q \rfloor$ (where ξ is the exploration fraction, e.g., 0.15) are selected with the lowest predicted $\hat{f}$ (i.e., most promising according to the surrogate).
  - The remaining $q_x = q - q_e$ are selected for diversity: these are the candidates with the largest minimum distance to any point in the archive (i.e., most novel).
- *Evaluation and Replacement:* Only these q selected candidates are evaluated with the true objective function, and their results are used to replace the invalid offspring in the population.

This surrogate screening strategy, as implemented in `GAWorker.py`, allows the algorithm to balance exploitation (focusing on candidates likely to improve the objective) and exploration (sampling novel regions of the search space), while reducing the number of expensive function evaluations per generation.

Algorithm 7 Surrogate screening of invalid offspring

```
Require: target evaluate count $q$, pool factor $\rho > 1$, history $\{(\boldsymbol{x}^{(j)}, y_j)\}$
1: build pool $\mathcal{U}$ by cloning/mutating/crossover until $|\mathcal{U}| = \lceil \rho q \rceil$
2: compute $\hat{f}(\cdot)$ by $k$NN in $[0,1]^d$; sort $\mathcal{U}$ ascending
3: $q_e \leftarrow \lfloor (1-\xi) q \rfloor$ exploit, $q_x \leftarrow q - q_e$ explore
4: choose first $q_e$ by lowest $\hat{f}$; choose $q_x$ by novelty (max min-distance)
5: evaluate chosen; replace invalid offspring accordingly
```

Surrogate decision.



7 Complete Algorithm

Algorithm 8 GAWorker main loop

```
1: initialize seeding method and population $\mathcal{P}_0$ (Algorithm 1); evaluate all
2: for $t = 1$ to $T$ do
3:     if paused/aborted: break
4:     select controller by decision tree; update $(p_{\text{cx}}, p_{\text{mut}}, N)$; resize if needed
5:     run one generation (selection, crossover, mutation, repair)
6:     evaluate invalid offspring; if surrogate enabled and ready, screen first
7:     update best-so-far, statistics $(\mu, \sigma, cv, D)$, success rate, EMAs
8:     apply adaptive heuristics or controller updates; log metrics
9:     if $\min f \le \varepsilon$: break
10: end for
11: return best individual and full metrics; compute final FRF and report
```

8 Neural Seeding (optional)

An ensemble surrogate is trained incrementally from $(\boldsymbol{x}, f(\boldsymbol{x}))$ pairs. Acquisition functions include UCB and EI. Given $\beta \in [\beta_{\min}, \beta_{\max}]$ and exploration fraction ε (optionally adapted with stagnation), propose a pool and select by acquisition and diversity; newly evaluated samples augment the training set.

8.1 Recommended Seeding–Surrogate–Adaptation Recipes

1. **LHS + kNN surrogate + ML–UCB**
   - Seeding: `lhs`. Surrogate: `use_surrogate=true`, `surrogate_k=7`, `surrogate_pool_factor=2`
     `surrogate_explore_frac=0.15`.
   - Adaptive: `use_ml_adaptive=true`, `ml_ucb_c=0.6`, `ml_adapt_population=true`, `pop_min`, $\approx 0.75N$, `pop_max`, $\approx 1.25N$.
   - Rationale: LHS provides stratified coverage that stabilizes kNN predictions; UCB balances exploration/exploitation with low regret, adapting rates and population to improve improvement-per-evaluation.

2. **Sobol + kNN surrogate (explorative) + heuristic adaptive rates**
    - Seeding: `sobol`. Surrogate: `k=5`, `pool_factor=2.0`, `explore_frac=0.30`.
    - Adaptive: `adaptive_rates=true`, `stagnation_limit=5`, diversity-driven increase in mutation when coefficient of variation is low.
    - Rationale: Low-discrepancy Sobol seeds reduce initial clustering; higher surrogate exploration mitigates model bias on multimodal landscapes; diversity-based heuristics sustain global search.

3. **Memory seeding + kNN surrogate + RL controller**
    - Seeding: `memory` (replay + jitter + explore). Surrogate: `k=5`, `pool_factor=3.0`, `explore_frac=0.20`.
    - Adaptive: `use_rl_controller=true`, `rl_alpha=0.1`, `rl_gamma=0.9`, `rl_epsilon=0.2`, `rl_epsilon_decay=0.95`, `pop_min`, $\approx 0.75N$, `pop_max`, $\approx 1.25N$.
    - Rationale: Warm starts exploit prior data to cut cold-start cost; RL learns effective rate/pop moves from rewards shaped by improvement, time, and diversity, reducing wasted evaluations.

4. **Neural seeding (UCB, epsilon-adaptive) + kNN surrogate + ML–UCB**
    - Seeding: `neural`, `neural_acq_type=ucb`, `neural_adapt_epsilon=true` (`eps_min=0.05`, `eps_max=0.30`), `neural_pool_mult=3.0`.
    - Surrogate: `k=5`, `pool_factor=2.0`, `explore_frac=0.15`. Adaptive: `use_ml_adaptive=true`, `ml_ucb_c=0.6`, `ml_adapt_population=true`.
    - Rationale: Model-guided seeds (UCB) deliver high-utility, diverse starts; epsilon adapts to stagnation; the bandit tunes cx/mut/pop to maximize improvement-per-cost while the surrogate prunes weak offspring.

5. **Best-of-pool QMC seeding + kNN surrogate (exploit-heavy) + heuristic adaptive rates**
    - Seeding: `best`, `best_pool_mult=5.0`, `best_diversity_frac=0.20`.
    - Surrogate: `k=7`, `pool_factor=2.0`, `explore_frac=0.10`. Adaptive: `adaptive_rates=true`, `stagnation_limit=4`.
    - Rationale: A broader QMC pool evaluated with the true objective yields a strong and diverse initial population; then a more exploitative surrogate accelerates convergence, while diversity-triggered rate changes maintain search breadth.