Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Physics-Informed Machine Learning Surrogates for Phase-Locked Loop Lock-Time Prediction, Validated Against Circuit-Level SPICE

Swarnam Nitnaware
Indian Institute of Technology Roorkee
Mayank Goswami
mayankfph@iitr.ac.in

Indian Institute of Technology Roorkee

Research Article

Keywords: phase-locked loop, lock time, surrogate model, physics-informed machine learning, residual learning, SPICE, ngspice

Posted Date: November 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11653042/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Physics-Informed Machine Learning Surrogates for Phase-Locked Loop Lock-Time Prediction, Validated Against Circuit-Level SPICE

Swarnam Nitnaware[1] and Mayank Goswami,[*1,2]

[1]Department of Physics, IIT Roorkee
[2]MF School of AI and DS, IIT Roorkee
mayank.goswami@ph.iitr.ac.in

Abstract

Lock time is a key specification of a phase-locked loop (PLL), but finding it by transient circuit simulation is slow, because the settling time is orders of magnitude longer than the reference period. We study machine-learning surrogates that predict the lock time of a third-order charge-pump PLL directly from its design parameters. Models are trained on 4,665 designs from a fast behavioral simulator and then validated on 80 unseen designs simulated in ngspice with a switched tri-state phase-frequency detector. The textbook lock-time estimate has a median error of 132% against ngspice. Physics-derived features and a residual formulation, in which the model learns only the correction to an improved analytical estimate, make the surrogate far more data-efficient: with 50 training designs it already outperforms a raw-feature model trained on all 4,665. For loop bandwidths below $0.02\,f_{\mathrm{ref}}$ (51 designs), the best models reach an RMSE of about 0.06 in $\log_{10}$ lock time (median error 0.03–0.04, roughly 7–10%), about 8,300× faster per query than ngspice. Above this bandwidth every model, including the training simulator itself, degrades, which we attribute to PFD sampling effects that the behavioral model omits. We report this limitation and the other limits of the study explicitly.

Keywords: phase-locked loop, lock time, surrogate model, physics-informed machine learning, residual learning, SPICE, ngspice

1 Introduction

1.1 Problem and motivation

Phase-locked loops (PLLs) generate clean, high-frequency clocks from a slow, accurate reference, and they appear in nearly every communication and computing system. When the target frequency changes, the loop needs a finite time to settle. This *lock time* is a headline specification of frequency synthesizers, because it limits how fast a system can hop between channels or wake from a low-power state.

Designers normally obtain lock time from a long transient simulation in a circuit simulator such as SPICE [11]. The simulation is expensive because the reference period is on the order of nanoseconds while settling takes tens of microseconds or more, so each run needs a very large number of time steps. A realistic design study must repeat this over many component

*Corresponding author: mayank.goswami@ph.iitr.ac.in

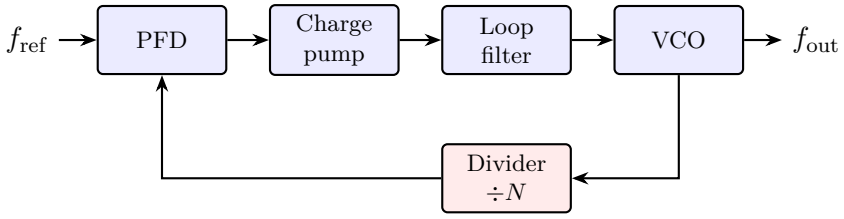

Figure 1: Block diagram of the charge-pump PLL.

values, process, voltage and temperature corners, and frequency steps. Closed-form estimates exist [1–3], but they come from linear loop theory and ignore both the size of the frequency step and the saturation of the charge pump, so they can be off by large factors.

This creates a need for a fast predictor of lock time that is more accurate than the textbook formula, and whose accuracy is established against an independent, more realistic simulation rather than against the data it was trained on.

1.2 Objectives

1. Build a surrogate that predicts PLL lock time from eight design parameters.
2. Quantify how much physics-derived features and a residual formulation help, in accuracy and in the number of training simulations required.
3. Validate the surrogate against circuit-level ngspice on designs it has never seen, and report the behavioral-to-circuit gap honestly.
4. Measure the computational saving, including the cost of generating training data.

1.3 Contributions

We (i) train on a cheap behavioral simulator and test on an independent ngspice circuit with a switched PFD; (ii) compare raw-feature, physics-feature and residual models, with an improved analytical formula as a fair baseline; (iii) show, with a learning curve scored against ngspice, that physics features make the surrogate data-efficient; and (iv) give a bandwidth-resolved account of where the behavioral model and the circuit disagree. Machine learning for analog circuits is an established area; we do not claim the idea is new, and we state the scope of our claims in Section 8.

2 Theory

2.1 The charge-pump PLL

Figure 1 shows the loop we study: a phase-frequency detector (PFD), a charge pump delivering current I_{cp}, a passive loop filter, a voltage-controlled oscillator (VCO) with gain K_{vco} (Hz/V), and a divide-by-N block in the feedback path, with reference frequency f_{ref}. The PFD compares the divided output with the reference. The charge pump converts that phase error into current pulses that charge or discharge the loop filter. The filter smooths them into the VCO control voltage.

The loop filter is the passive network of Fig. 2: the charge-pump current i_{cp} feeds a node with capacitor C_2 to ground, and in parallel a series branch of R_1 and C_1. Its node voltage v_2 is the VCO control voltage.

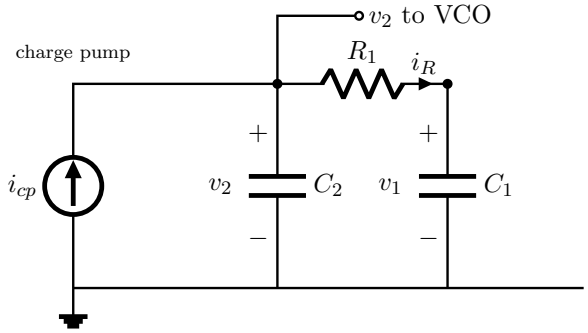

Figure 2: Third-order passive loop filter (R_1, C_1, C_2) driven by the charge-pump current.

2.2 Linear model

With phase-detector gain $K_{pd} = I_{cp}/2\pi$ (A/rad) and VCO gain $K_v = 2\pi K_{vco}$ (rad/s/V), the natural frequency and damping ratio are

$$\omega_n = \sqrt{\frac{K_{pd}K_v}{N(C_1 + C_2)}}, \qquad \zeta = \tfrac{1}{2} R_1 C_1 \, \omega_n. \tag{1}$$

With $\tau_z = R_1 C_1$ and $\tau_p = R_1 C_1 C_2/(C_1 + C_2)$, the open-loop gain is

$$L(s) = \frac{\omega_n^2 \, (1 + s\tau_z)}{s^2 \, (1 + s\tau_p)}. \tag{2}$$

Its unity-gain (crossover) frequency ω_c is the loop bandwidth, and the phase margin is $\mathrm{PM} = \arctan(\omega_c \tau_z) - \arctan(\omega_c \tau_p)$. The ratio $f_c/f_{\mathrm{ref}} = \omega_c/(2\pi f_{\mathrm{ref}})$ will turn out to be the most important variable in this study.

2.3 Lock definition and textbook estimate

A design is *locked* when the normalized frequency error stays below $\epsilon_f = 10^{-3}$ and the phase error stays below 0.3 rad for the remainder of the run. Lock time is the instant after the last violation of this band, and a run counts as locked only if the last 10% of the simulated window lies inside the band. Using the last violation, not the first entry, prevents a ringing response that leaves the band again from being counted as locked. The definition was fixed before any model was trained.

For a decay rate $\sigma = \zeta\omega_n$ (if $\zeta \le 1$) or $\sigma = \omega_n(\zeta - \sqrt{\zeta^2 - 1})$ (if $\zeta > 1$), the textbook estimate is

$$t_{\mathrm{est}} = \frac{\ln(1/\epsilon_f)}{\sigma}. \tag{3}$$

It assumes the loop starts at 100% error. If the step produces an initial normalized frequency error $f_0 = K_{vco}\Delta V/(N f_{\mathrm{ref}})$, an *improved* estimate accounts for the shorter distance to the band:

$$t_{\mathrm{imp}} = t_{\mathrm{est}} \, \frac{\ln(f_0/\epsilon_f)}{\ln(1/\epsilon_f)}. \tag{4}$$

We use both as baselines, so the machine-learning gain is measured against a physics formula that already uses the step size.

2.4 Charge-pump saturation

The phase detector and charge pump deliver a current proportional to the phase error only up to one reference cycle; beyond $\pm 2\pi$ the current saturates at $\pm I_{cp}$. For large frequency steps the

loop therefore slews at a limited rate and locks more slowly than linear theory predicts. This nonlinearity, absent from Eq. (4), is the main reason a learned correction can help.

2.5 Sampling and the validity range

The continuous-time model treats the PFD and charge pump as a smooth, averaged current. A real PFD acts once per reference cycle, which adds a delay of roughly half a reference period [1]. This is negligible when $f_c \ll f_{\text{ref}}$, but it erodes phase margin as f_c/f_{ref} grows. The continuous-time model is therefore restricted to $f_c/f_{\text{ref}} < 0.1$ in this work, and Section 5 shows that accuracy against a switched circuit degrades well before that limit.

3 Related Work

PLL analysis has a long tradition, from the charge-pump loop theory of Gardner [1] to standard design texts [2–4]. Behavioral simulation is also long established as a faster alternative to transistor-level transient simulation of PLLs and related loops [5]; our Python simulator is a minimal example of this class.

Machine learning has been applied to analog circuit design, mostly for sizing and optimization: Bayesian optimization of analog circuits [6], reinforcement learning for sizing [7], and graph-neural-network-based reinforcement learning with transfer across technologies [8]. These works target design optimization more than surrogate models of a single dynamic metric such as lock time.

Our residual formulation is related to two broader ideas. Physics-informed neural networks embed governing equations in the learning problem [9]; we instead feed physics-derived quantities as features and model the residual to an analytical estimate. Multi-fidelity modeling combines a cheap approximation with expensive data through a learned discrepancy term [10], which parallels our use of a cheap simulator for training and a circuit simulator for validation. We did not perform a systematic literature search, so we make no claim of priority for any element of the approach; the contribution is the end-to-end validation, the data-efficiency analysis against circuit-level results, and the bandwidth-resolved account of the behavioral-to-circuit gap.

4 Method

4.1 Design space and sampling

Eight parameters were sampled with Latin Hypercube sampling [12], all on a logarithmic scale because they span decades: $I_{cp} \in [10\,\mu\text{A}, 1\,\text{mA}]$, $R_1 \in [0.5, 20]\,\text{k}\Omega$, $C_1 \in [20\,\text{pF}, 2\,\text{nF}]$, $C_2/C_1 \in [0.02, 0.3]$, $K_{vco} \in [20, 500]\,\text{MHz/V}$, $N \in [20, 500]$, $f_{\text{ref}} \in [1, 50]\,\text{MHz}$, and the control-voltage step $\Delta V \in [0.05, 1]\,\text{V}$.

4.2 Behavioral simulator

The behavioral model integrates three states, the phase error θ, the voltage v_1 on C_1, and the VCO control voltage v_2 on C_2, all as deviations from the locked value:

$$\dot{\theta} = -K_v v_2 / N, \tag{5}$$

$$i_{cp} = I_{cp}\,\mathrm{clip}(\theta/2\pi,\, -1,\, 1)\,, \qquad i_R = \frac{v_2 - v_1}{R_1}, \tag{6}$$

$$\dot{v}_1 = \frac{i_R}{C_1}, \qquad \dot{v}_2 = \frac{i_{cp} - i_R}{C_2}. \tag{7}$$

The initial condition is $v_1 = v_2 = -\Delta V$, which represents a frequency step of $K_{vco}\Delta V$. The system was integrated with the LSODA solver [13] in SciPy [14] over a window of $\max(8t_{\mathrm{est}},\, 100/f_{\mathrm{ref}})$, sampled at 6,000 points, and the lock definition of Section 2.3 was applied.

4.3 Dataset

Of 6,000 sampled designs, 4,916 satisfied the validity filter $f_c/f_{\mathrm{ref}} \le 0.1$, and 4,901 of those locked (99.7%). We removed 236 designs with $f_0 \le 10^{-3}$, which start inside the lock band and have $t_{\mathrm{lock}} = 0$, so that $\log t_{\mathrm{lock}}$ is undefined. This left 4,665 training rows. Lock times span several decades with a median of 33 μs, so the regression target is $y = \log_{10} t_{\mathrm{lock}}$. Because almost every valid design locks, we did not pursue a lock/no-lock classifier.

4.4 Features and models

Three feature sets were compared. **A (raw)**: the eight parameters, $\log_{10}$-scaled. **B (physics)**: A plus $\log_{10}\zeta$, phase margin, $\log_{10}(f_c/f_{\mathrm{ref}})$, $\log_{10} f_0$, $\log_{10} t_{\mathrm{est}}$ and $\log_{10} t_{\mathrm{imp}}$. **C (residual)**: the inputs of B, but the model predicts $y - \log_{10} t_{\mathrm{imp}}$, and the final prediction adds $\log_{10} t_{\mathrm{imp}}$ back.

Four regressors from scikit-learn [16] and XGBoost [17] were used: ridge regression with cross-validated regularization, a random forest (300 trees, minimum leaf size 2), gradient-boosted trees (600 trees, depth 5, learning rate 0.05, subsample and column sample 0.8), and a multilayer perceptron (two hidden layers of 128 units, early stopping). For ridge regression, sets B and C are mathematically equivalent because $\log_{10} t_{\mathrm{imp}}$ is already an input, which serves as a check of the residual code.

4.5 Evaluation protocol

Two splits were used against the behavioral simulator: a random 80/20 split, and an extrapolation split that trains on the 85% of designs with the smallest f_0 and tests on the largest 15%, where slewing dominates. Errors are computed in $\log_{10}$ lock time: RMSE, median absolute error, 90th-percentile absolute error (P90), and median absolute percentage error (MAPE) in linear lock time. An error of e in $\log_{10}$ is a multiplicative factor of 10^e. R^2 is not compared across splits because the extrapolation test set has a narrower spread.

4.6 Circuit-level validation in ngspice

The independent test set is 80 designs drawn with a different random seed, so none overlap the training set. They were simulated in ngspice 42 with a switched circuit: a behavioral reference clock, a tri-state PFD built from two D flip-flops and a reset gate (XSPICE digital models), a current-switching charge pump, the real R_1–C_1–C_2 filter, and a behavioral VCO and divider that integrate the control voltage into phase. The initial voltage offset matches the behavioral

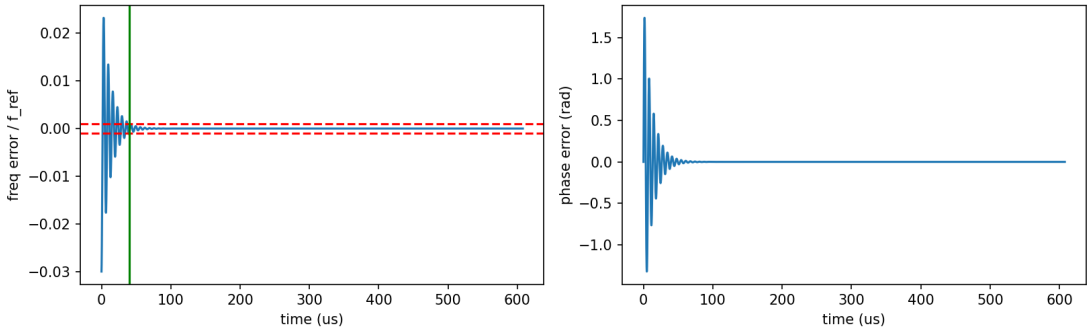

Figure 3: Behavioral simulation of one design: frequency error (left) and phase error (right). Dashed lines mark the ±0.1% lock band; the green line is the detected lock time.

model. The PFD pulses cause ripple on the control voltage, so the control voltage is averaged over one reference period before the same lock definition is applied. This is a centered moving average, so it adds no systematic delay, and its smoothing effect is at most about half a reference period.

To keep each ngspice run short, only designs with a behavioral lock time under 150 μs that passed the validity filter and had $f_0 > 10^{-3}$ were kept, and each ngspice window was six times the behavioral lock time. All validation conclusions therefore apply to this range only.

4.7 Implementation

All experiments ran on Google Colab (CPU) with Python, NumPy [15], SciPy, scikit-learn, XGBoost and ngspice. Random seeds were 0 (pilot run), 1 (training designs) and 7 (ngspice validation designs), and model seeds were fixed at 0.

5 Results

5.1 Behavior of the simulator and the textbook baseline

Figure 3 shows a typical low-damping design: the frequency error rings and decays into the lock band, and the detected lock time is well below the textbook estimate of about 76 μs for that design. Across the whole dataset (Fig. 4), the textbook estimate has a median $|\log_{10}\text{error}|$ of 0.35, a 90th-percentile error of 0.82, and $R^2 = 0.57$ on the log scale. It mostly overestimates lock time, because it ignores that the loop starts close to the band for small steps, while for the largest steps slewing makes the real loop slower, so the two error sources partly oppose each other.

5.2 Results against the behavioral simulator

Table 1 summarizes selected models on both splits. On the random split, the best residual model reduces the RMSE of the improved textbook from 0.260 to 0.071, and its 90th-percentile error corresponds to a factor of about 1.24. Tree models degrade more than the MLP on the extrapolation split (XGBoost 0.072 to 0.149; MLP 0.071 to 0.101), consistent with trees being unable to predict outside the range of values seen in training. The MLP and XGBoost are effectively tied on the random split, so we do not rank them.

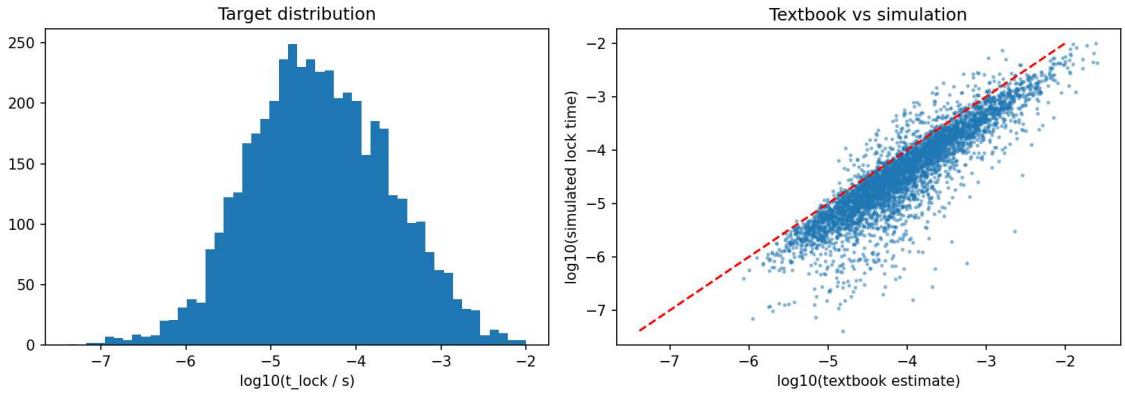

Figure 4: Target distribution (left) and textbook estimate versus simulated lock time (right). The dashed line is the ideal.

Table 1: Error in $\log_{10}$ lock time against the behavioral simulator (selected models).

	Random split		Extrapolation	
Model	RMSE	P90	RMSE	P90
Textbook	0.553	0.771	0.232	0.431
Improved textbook	0.260	0.434	0.220	0.383
XGBoost, A (raw)	0.170	0.219	0.146	0.243
MLP, A (raw)	0.119	0.155	0.123	0.201
XGBoost, B (physics)	0.089	0.128	0.150	0.230
MLP, B (physics)	0.096	0.141	0.119	0.194
RandForest, C (residual)	0.084	0.104	0.168	0.261
XGBoost, C (residual)	0.072	0.097	0.149	0.227
MLP, C (residual)	**0.071**	**0.092**	**0.101**	**0.135**

5.3 Validation against ngspice

Figure 5 shows a single design simulated both ways. The curves nearly coincide, with a lock time of 12.13 μs in the behavioral model and 14.19 μs in ngspice (+17%); the small ripple on the ngspice curve comes from the switching PFD.

Table 2 gives the results on the 80 ngspice designs. The row "Behavioral simulator" evaluates the training simulator itself against ngspice and sets a floor for any model trained only on its data. Three observations follow. First, physics-feature models clearly outperform raw-feature models (RMSE 0.224–0.234 versus 0.245–0.287), and raw-feature models are no better than the improved textbook. Second, the best models reach the floor set by the behavioral simulator (about 0.23), so they reproduce their training simulator closely and the remaining error is the behavioral-to-circuit gap. Third, with only 80 designs, differences of about 0.01 among the physics and residual models are within noise, so we do not claim that residual learning beats plain physics features against ngspice, even though it helps on the behavioral extrapolation split. Figures 6 and 7 show the predictions and the ranking.

5.4 Where the behavioral model and the circuit disagree

The median disagreement between the behavioral simulator and ngspice is small (0.023 in $\log_{10}$), but the 90th percentile is large (0.40), so a minority of designs dominates the RMSE. The absolute gap correlates with f_c/f_{ref} with a Spearman coefficient of +0.74; no other variable has $|\rho|$ above 0.4. Table 3 splits the designs into terciles of f_c/f_{ref}: the signed gap is positive in

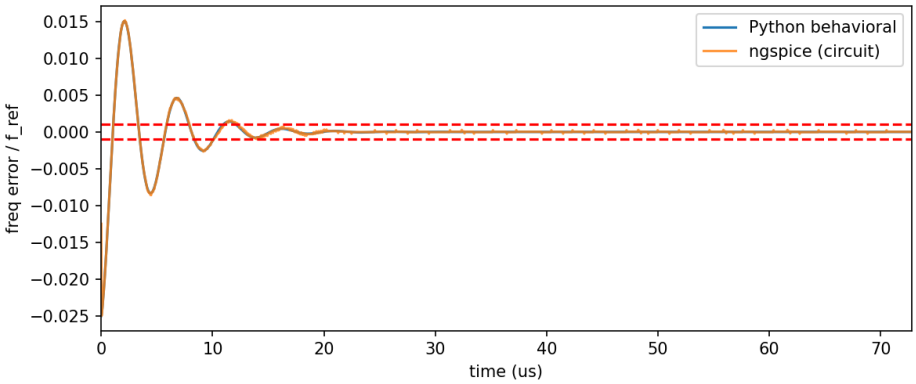

Figure 5: One design simulated in the behavioral model and in ngspice (frequency error versus time).

Table 2: Error in $\log_{10}$ lock time against ngspice (80 designs).

Model	RMSE	Median	P90	MAPE (%)
Textbook	0.499	0.393	0.691	132.1
Improved textbook	0.272	0.084	0.564	17.8
Behavioral simulator (floor)	0.232	0.023	0.403	5.2
XGBoost, A (raw)	0.263	0.121	0.429	26.8
MLP, A (raw)	0.245	0.083	0.438	18.1
XGBoost, B (physics)	0.229	0.048	0.338	10.4
MLP, B (physics)	0.228	0.065	0.372	14.2
RandForest, C (residual)	0.229	0.041	0.344	9.7
XGBoost, C (residual)	0.228	0.049	0.348	11.7
MLP, C (residual)	0.224	0.052	0.384	11.8

every tercile, so ngspice is systematically slower, and the effect grows with bandwidth. Table 4 scores the models separately at $f_c/f_{\text{ref}} = 0.02$.

Below 0.02, the residual models cut the RMSE of the improved textbook by about a factor of three (0.178 to about 0.06), with a median error of 0.03–0.04, which is roughly 7–10% in lock time. They are slightly worse than the behavioral simulator itself (0.058–0.063 versus 0.046), as expected for a model trained on that simulator. Above 0.02, no model is meaningfully better than the improved textbook, including the behavioral simulator, so the error there is a limitation of the training data and not of the learning algorithm. The overall RMSE of 0.22–0.23 averages two different regimes and should not be read alone. The 0.02 threshold was chosen after inspecting this data, so the split is exploratory.

A plausible explanation is the half-period PFD sampling delay discussed in Section 2.5. *We have not tested this mechanism directly*, for example by adding the delay to the behavioral model, so it remains a hypothesis supported by the correlation.

5.5 Data efficiency

Figure 8 shows the RMSE against ngspice versus training-set size for XGBoost, averaged over three random training subsets. The physics-and-residual model trained on only 50 designs (RMSE about 0.25) already outperforms the raw-feature model trained on all 4,665 designs (about 0.27), which plateaus at the level of the improved textbook. The physics model approaches the behavioral floor at about 1,000 designs, after which more data helps little because the remaining error is the behavioral-to-circuit gap. The error bars show variation across training subsets and do not include the uncertainty from the 80-design validation set, and the

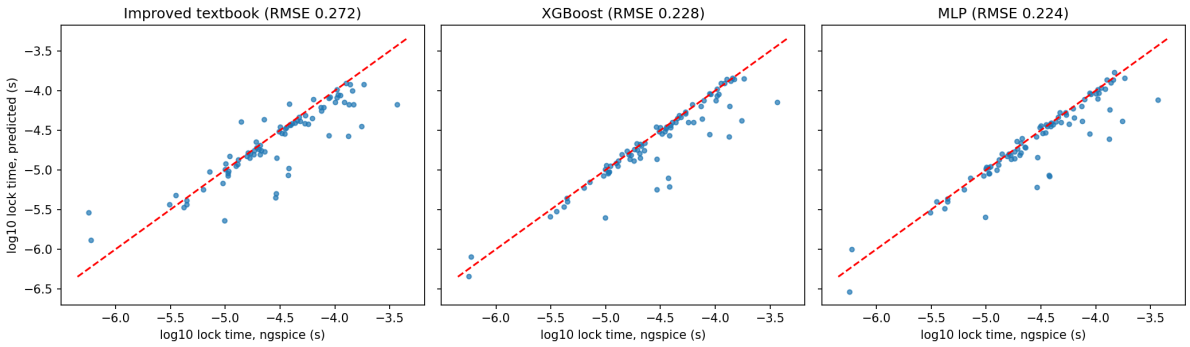

Figure 6: Predicted versus ngspice $\log_{10}$ lock time for the improved textbook, XGBoost and MLP (residual models). The dashed line is the ideal.

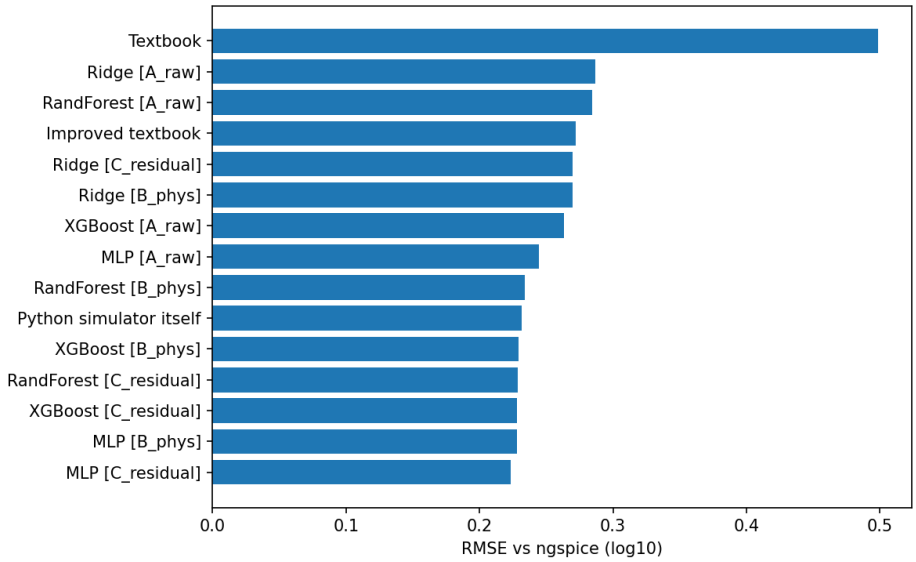

Figure 7: RMSE against ngspice for every baseline and model/feature combination (lower is better).

advantage over the improved textbook at 50–250 designs is modest.

5.6 Computational cost

The average ngspice run took 3.73 s per design on the validation set. Prediction takes 0.45 ms per design (0.42 ms of it computing the analytical features), a per-query speedup of about 8,300× relative to ngspice. The one-time cost of generating training data and fitting the model was 16.4 min, so the surrogate pays off after about 264 designs when ngspice would otherwise be used for every design: for 50 designs ngspice is cheaper overall, while for 5,000 designs the surrogate is about 19 times cheaper. Relative to the behavioral simulator (about 0.2 s per design), the speedup is only about 400×. The ngspice time is measured only for designs with lock times under 150 μs, and the training cost was incurred with the behavioral simulator, not ngspice.

6 Discussion

Three conclusions follow. First, physics-derived features are what make the surrogate useful: raw-feature models never learn beyond the improved formula, while the same models given ζ, phase margin and the analytical estimates reach the accuracy of their training simulator

Table 3: Behavioral-to-ngspice gap by loop-bandwidth tercile ($\log_{10}$ units).

Tercile	f_c/f_{ref}	Median |gap|	P90 |gap|	Mean signed
Low ($n = 27$)	0.001–0.007	0.003	0.031	+0.012
Mid ($n = 26$)	0.007–0.021	0.014	0.114	+0.053
High ($n = 27$)	0.021–0.096	0.142	0.709	+0.256

Table 4: RMSE and median error in $\log_{10}$ lock time against ngspice, by bandwidth group.

	$f_c/f_{\text{ref}} < 0.02$ ($n = 51$)		≥ 0.02 ($n = 29$)	
Model	RMSE	Median	RMSE	Median
Improved textbook	0.178	0.072	0.385	0.104
Behavioral simulator	0.046	0.006	0.380	0.142
XGBoost, C	0.058	0.031	0.371	0.143
MLP, C	0.063	0.041	0.362	0.113

with far less data. Second, the surrogate is accurate against a more realistic circuit exactly where the behavioral model is, which is for loop bandwidths below about $0.02 f_{\text{ref}}$. Third, a surrogate trained on a simulator cannot be more trustworthy than that simulator, which is why independent circuit-level validation changes the conclusions: against the behavioral simulator alone, the best residual model improved the RMSE by 3.6× over the improved textbook, whereas against ngspice the overall improvement is only about 18%. Given the tail errors, the surrogate is best used to screen and rank many candidate designs, with the best few confirmed in circuit simulation, not for signoff.

7 Conclusion

We trained physics-informed surrogates for PLL lock time on a behavioral simulator and validated them against switched-circuit ngspice. For $f_c/f_{\text{ref}} < 0.02$ they predict lock time with a median error of about 7–10% at a per-query speedup of about 8,300× over ngspice, and they are far more data-efficient than raw-feature models. Above this bandwidth the behavioral-to-circuit gap dominates every model.

8 Limitations

Several constraints bound the scope of this validation. First, the validation covers only lock times under 150 μs across a set of 80 validation designs; consequently, close rankings among the models are not statistically meaningful. Furthermore, all evaluations rely on a single ngspice circuit composed of ideal behavioral blocks, omitting transistor-level dynamics, noise, and mismatch effects, as well as process, voltage, or temperature (PVT) corners. As a result, the models naturally inherit the blind spots of the underlying behavioral simulator.

Additional limitations pertain to analytical choices, empirical thresholds, and literature scope. The learning curve evaluation uses XGBoost only, assessed with three random subsets per dataset size. Furthermore, the 0.02 bandwidth threshold was selected after inspecting the data and requires confirmation on a fresh validation set. Similarly, the proposed sampling-delay explanation remains a hypothesis rather than a tested mechanism. Finally, because the literature review was not systematic, no priority claim is made.

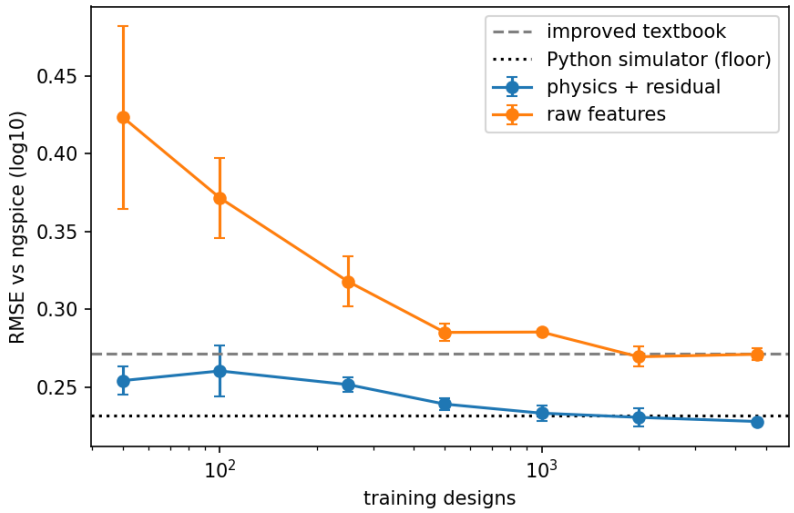

Figure 8: RMSE against ngspice versus number of training designs (XGBoost, mean ± s.d. over three subsets). Dashed: improved textbook; dotted: behavioral simulator floor.

Future work

(i) Add the half-period sampling delay to the behavioral simulator and test whether the high-bandwidth gap closes; (ii) train a small residual correction on 50–100 ngspice runs; (iii) use uncertainty-driven active learning to choose which designs to simulate; (iv) repeat the validation on a fresh ngspice set to confirm the bandwidth threshold; and (v) validate at transistor level and across process, voltage and temperature corners.

Code and data availability

The code, the training data (4,665 behavioral-simulator designs, `dataset_clean.csv`), the testing data (80 ngspice designs, `spice_results.csv`), the result tables and the figures will be released at `https://colab.research.google.com/drive/1ZrEYJtNfqG_YH7LZiloUSLcLNMVHnpgY`. Random seeds are listed in Section 4.7.

Video

`https://youtu.be/D-Nk7ybctSA`

==========

References

[1] F. M. Gardner, "Charge-pump phase-lock loops," *IEEE Trans. Commun.*, vol. 28, no. 11, pp. 1849–1858, Nov. 1980.

[2] B. Razavi, *RF Microelectronics*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2011.

[3] R. E. Best, *Phase-Locked Loops: Design, Simulation, and Applications*, 6th ed. New York, NY, USA: McGraw-Hill, 2007.

[4] D. Banerjee, *PLL Performance, Simulation, and Design*, 4th ed. Dog Ear Publishing, 2006.

[5] M. H. Perrott, "Fast and accurate behavioral simulation of fractional-N frequency synthesizers and other PLL/DLL circuits," in *Proc. 39th Design Automation Conf. (DAC)*, 2002.

[6] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou, "An efficient Bayesian optimization approach for automated optimization of analog circuits," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 6, 2018.

[7] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. Design, Automation & Test in Europe (DATE)*, 2020.

[8] H. Wang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. 57th ACM/IEEE Design Automation Conf. (DAC)*, 2020.

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019.

[10] M. C. Kennedy and A. O'Hagan, "Predicting the output from a complex computer code when fast approximations are available," *Biometrika*, vol. 87, no. 1, pp. 1–13, 2000.

[11] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Univ. California, Berkeley, Tech. Rep. ERL-M520, 1975.

[12] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[13] L. Petzold, "Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations," *SIAM J. Sci. Stat. Comput.*, vol. 4, no. 1, pp. 136–148, 1983.

[14] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[15] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020.

[16] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 785–794.