An Empirical Investigation of Initialization Strategies for Kolmogorov–Arnold Networks

Spyros Rigas [1] **Dhruv Verma** [2] **Georgios Alexandridis** [1] **Yixuan Wang** [2]

Abstract

Kolmogorov–Arnold Networks (KANs) are a recently introduced neural architecture that use trainable activation functions instead of fixed ones, offering greater flexibility and interpretability. Although KANs have shown promising results across various tasks, little attention has been given to how they should be initialized. In this work, we explore alternative initialization strategies, including two variance-preserving methods based on classical ideas and an empirical power-law approach with tunable exponents. Using function fitting as a small-scale testbed, we run a large grid search over architectures and initialization settings. We find that power-law configurations consistently outperform the standard baseline initialization across all architectures. The variance-preserving methods tend to underperform on smaller models but outperform the baseline as networks grow deeper and wider, though they still do not match the performance of power-law initialization. Overall, our results highlight initialization as an important yet underexplored aspect of KANs and point to several directions for future work.

1. Introduction

Kolmogorov–Arnold Networks (KANs) (Liu et al., 2025) have recently emerged as an alternative backbone architecture to Multilayer Perceptrons (MLPs), drawing inspiration from the Kolmogorov–Arnold representation theorem (Kolmogorov, 1957) in a manner analogous to how the learning of MLPs relies on universal approximation theorems. Unlike MLPs, which use fixed nonlinear activation functions and trainable linear layers, KANs comprise grid-dependent trainable activation functions. This provides them with flexibility in modeling complex nonlinear relationships, while requiring fewer and smaller layers. Since their introduction, KANs have found numerous applications, often surpassing the performance of their MLP-based counterparts (Yu et al., 2024; Poeta et al., 2024). There have been many notable results in scientific problem-solving domains, including function fitting and symbolic regression (Liu et al., 2024; Shukla et al., 2024), partial differential equations (PDEs) (Shukla et al., 2024; Rigas et al., 2024; Wang et al., 2025b) and operator learning (Abueidda et al., 2025; Shukla et al., 2024), among other applications (Howard et al., 2024; Kundu et al., 2024; Kashefi, 2025).

Beyond these benchmarks, there has also been significant progress in the theoretical understanding of KANs (Zhang & Zhou, 2025; Alter et al., 2025; Wang et al., 2025a). However, one important theoretical and practical aspect that remains understudied pertains to their initialization strategies. Current literature mainly relies on the standard initialization method proposed in the introductory KAN paper (Liu et al., 2025), highlighting a clear gap and motivating an investigation into potentially more effective initialization approaches. Effective initialization is crucial, as a good "initial guess" for the network weights can significantly accelerate training (Mishkin & Matas, 2016; Skorski et al., 2021) and prevent early saturation of hidden layers (Glorot & Bengio, 2010). However, despite extensive research into initialization methods for MLP-based architectures, these results cannot be directly applied to KANs. Furthermore, even within MLP-based architectures, initialization methods often require separate consideration depending on the specific architecture design (Huang et al., 2020), activation function (He et al., 2015), or even on a complete case-by-case basis (Skorski et al., 2021).

This preliminary work explores initialization strategies for KANs. We focus on function fitting tasks, which are well-suited for small-scale experiments that can be run efficiently on limited computational resources, while noting that our broader aim

[1]Department of Digital Industry Technologies, School of Science, National and Kapodistrian University of Athens (NKUA), 344 00 Psachna, Greece [2]Applied and Computational Mathematics, California Institute of Technology, Pasadena, CA 91125, United States of America. Correspondence to: Spyros Rigas <spyrigas@uoa.gr>, Yixuan Wang <roywang@caltech.edu>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

also includes extending this study to forward PDE problems. We draw conceptual parallels between initialization methods in MLPs and KANs, investigating variance-preserving schemes inspired by LeCun initialization (LeCun et al., 1998) and batch normalization (Ioffe & Szegedy, 2015). Moreover, recognizing that theoretical frameworks may not always align with empirical performance (Mishkin & Matas, 2016), we further propose an empirical power-law-based initialization approach. This work aspires to establish foundational insights for further research into initialization strategies for KANs.

2. Background

2.1. Kolmogorov–Arnold Networks

Within the standard formalism, the output, $\mathbf{y} \in \mathbb{R}^{n_{\text{out}}}$, of a KAN layer is related to its input, $\mathbf{x} \in \mathbb{R}^{n_{\text{in}}}$, via:

$$y_j = \sum_{i=1}^{n_{\text{in}}} \left(r_{ji} \cdot R\left(x_i\right) + c_{ji} \sum_{m=1}^{G+k} b_{jim} \cdot B_m\left(x_i\right) \right), \quad j = 1, \ldots, n_{\text{out}}, \tag{1}$$

where r_{ji}, c_{ji} and b_{jim} are the layer's trainable parameters, $R(x)$ corresponds to a residual function, typically chosen as the SiLU, i.e., $R(x) = x\left(1 + e^{-x}\right)^{-1}$, and $B_m\left(x\right)$ denotes a univariate spline basis function of order k, defined on a grid with G intervals. For each of the layer's trainable parameters, the original KAN formulation initializes the scaling weights as $c_{ji} = 1$, the residual weights r_{ji} using Glorot initialization (Glorot & Bengio, 2010), and the basis weights b_{jim} from a normal distribution with zero mean and small standard deviation, typically set to $\sigma = 0.1$.

2.2. Related Work

In the existing KAN literature, initialization strategies have only been explored in certain KAN variants (Guilhoto & Perdikaris, 2025), while the standard KAN architecture has not yet received dedicated attention in this regard. A natural starting point for studying initialization is to follow the corresponding historical developments in MLP-based architectures, beginning with the scheme proposed by LeCun (LeCun et al., 1998), which ensures that the variance of activations remains stable across layers, preventing progressive vanishing or explosion. Alternatively, methods such as batch normalization (Ioffe & Szegedy, 2015) modify the architecture itself to maintain stable activation distributions throughout the network, reducing the dependence on careful initialization. Of course, it remains unclear whether such strategies directly transfer to KANs, given their architectural differences, thus motivating the investigation presented in this work.

3. Methodology

3.1. Proposed Initializations

Since the three different weight types in a KAN layer are independent, we may initialize the scaling weights c_{ji} to 1 and focus exclusively on the initialization of the residual weights r_{ji} and the basis weights b_{jim}. We assume that these weights are drawn from zero-mean distributions with standard deviations σ_r and σ_b, respectively. To determine suitable values for σ_r and σ_b, we follow the principle of variance preservation proposed by LeCun (LeCun et al., 1998), which stipulates that the variance of layer outputs should match that of the inputs, thereby avoiding signal amplification or attenuation across layers. Assuming statistical independence among terms and an equal contribution to the variance from each of the $(G + k + 1)$ terms inside the first summand of Eq. (1), we derive the following expressions for the standard deviations[1]:

$$\sigma_r = \sqrt{\frac{\operatorname{Var}\left(x_i\right)}{n_{\text{in}}\left(G + k + 1\right)\mathbb{E}\left[R^2\left(x_i\right)\right]}}, \qquad \sigma_b = \sqrt{\frac{\operatorname{Var}\left(x_i\right)}{n_{\text{in}}\left(G + k + 1\right)\mathbb{E}\left[B_m^2\left(x_i\right)\right]}}. \tag{2}$$

If we further assume that each component of $\mathbf{x}$ is drawn from a uniform distribution $\mathcal{U}\left(-1, 1\right)$, as is often the case in tasks like function fitting or PDE solving, then all statistical quantities in Eq. (2) can be evaluated directly, except for the expectation of the squared spline basis functions applied to the inputs. Due to the dependence of these basis functions on the underlying grid, no general analytic expression exists for σ_b. This leads to two practical alternatives: one may either estimate

[1] See Appendix A for detailed derivations.

$\mathbb{E}\left[B_m^2\left(x_i\right)\right]$ numerically by sampling a large number of input points from $\mathcal{U}\left(-1,1\right)$ at initialization, or set the expectation value to unity by modifying the architecture of the KAN layer to use batch-normalized spline basis functions, defined as

$$\tilde{B}_m\left(x_i\right) = \frac{B_m\left(x_i\right) - \mathbb{E}\left[B_m\left(x_i\right)\right]}{\sqrt{\mathbb{E}\left[B_m^2\left(x_i\right)\right] - \mathbb{E}^2\left[B_m\left(x_i\right)\right]}}, \tag{3}$$

where the expectation values are computed over the current batch during each forward pass. We will refer to the former alternative as "LeCun–numerical" initialization, while the latter is referred to as "LeCun–normalized" initialization.

In addition to these two theory-driven initialization strategies, we also investigate an empirical approach based on a power-law scaling of the KAN layer's architectural parameters. Specifically, we initialize the weights such that their standard deviations follow the form

$$\sigma_r = \left(\frac{1}{n_{\text{in}}\left(G+k+1\right)}\right)^{\alpha}, \qquad \sigma_b = \left(\frac{1}{n_{\text{in}}\left(G+k+1\right)}\right)^{\beta}, \tag{4}$$

where α and β are tunable exponents selected from the set $\{0.25, 0.5, \ldots, 1.75, 2.0\}$. The motivation behind this empirical scheme is to perform a grid search over (α, β) configurations in order to identify trends or specific exponent pairs that consistently improve training speed and convergence, potentially revealing an effective heuristic for initializing KANs.

3.2. Experimental Setup

We evaluate our initialization strategies on function fitting tasks, using five distinct two-dimensional functions the formulas and implementation details of which are provided in Appendix B. KANs are trained for 2000 iterations, and performance is evaluated using both the final training loss and the relative L^2 error between the model and reference solutions. As a grid-search process, each setting is tested across architectures with 1–4 hidden layers and widths ranging from 2 to 64, and across grid sizes $G \in \{5, 10, 20, 40\}$. All experiments are repeated under five random seeds, except those involving the empirical power-law initialization, which are run under three seeds to reduce computational cost. We report the median result in terms of final training loss across runs. All training is performed in `JAX` (Bradbury et al., 2018) using the `jaxKAN` framework (Rigas & Papachristou, 2025), on a single NVIDIA GeForce RTX 4090 GPU.

4. Experimental Results

The grid search over (α, β) configurations and the architectural variations described in Section 3 results in the training of 123840 KAN model instances. After aggregating the five (or three) repeated runs per setting by their median outcome, this total reduces to 40320 representative results. From these, we retain only the best-performing (α, β) configuration per setting, yielding 1920 final results. Table 1 reports, for each target function and initialization scheme, the percentage of runs that outperform the baseline initialization described in Section 2.1. Comparisons are made with respect to both the final training loss and the relative L^2 error to the reference solution. We also report the percentage of runs for which both metrics improve simultaneously over the baseline.

Table 1. Percentage of runs, grouped by target function and initialization scheme, that outperform the baseline initialization in terms of final training loss, relative L^2 error, and both metrics simultaneously. The best performance for each case is shown in bold font.

	LeCun–numerical			LeCun–normalized			Power-law		
Function	Loss	L^2	Both	Loss	L^2	Both	Loss	L^2	Both
$f_1\left(x,y\right)$	18.75%	6.25%	1.04%	19.79%	11.46%	2.08%	**100.00%**	**100.00%**	**100.00%**
$f_2\left(x,y\right)$	14.58%	4.17%	0.00%	28.13%	9.38%	5.21%	**100.00%**	**100.00%**	**100.00%**
$f_3\left(x,y\right)$	12.50%	5.21%	0.00%	19.79%	11.46%	5.21%	**100.00%**	**100.00%**	**100.00%**
$f_4\left(x,y\right)$	25.00%	14.58%	8.33%	41.67%	26.04%	16.67%	**100.00%**	**94.79%**	**94.79%**
$f_5\left(x,y\right)$	26.04%	2.08%	0.00%	31.25%	6.25%	1.04%	**98.96%**	**96.88%**	**95.83%**

Table 1 presents the final aggregated results, demonstrating that for nearly all cases, there exists a power-law initialization that outperforms the baseline. In contrast, this is not consistently true for the LeCun-based initialization strategies. However,

the table masks several important qualitative trends. Notably, while the baseline initialization performs better for smaller architectures, the LeCun-based strategies - especially the normalized variant - tend to surpass the baseline as the network depth, width, and grid size, G, increase. In several cases, these improvements exceed two orders of magnitude. Another notable observation is that the network tends to achieve significantly lower final losses for (α, β) configurations where α is relatively small (e.g., near 0.25) and β is on the higher end of the studied spectrum (i.e., ≥ 1.5). For a more detailed breakdown of these trends and the full grid search results, we refer the reader to Appendix C.

To gain a better understanding of these trends, we compare all four initialization strategies on two representative settings: a "small" architecture ($G = 5$, two hidden layers with 8 neurons each) and a "large" architecture ($G = 20$, three hidden layers with 32 neurons each). To ensure consistency across benchmarks, we fix the power-law parameters to $\alpha = 0.25$ and $\beta = 1.75$. Training follows the setup from Section 3.2, with each experiment repeated five times. The resulting training loss curves, averaged across runs, are shown in Figure 1 along with the corresponding standard error. For the small architecture, the power-law initialization yields the lowest losses, while the baseline outperforms both LeCun-based methods. In contrast, for the larger architecture, the power-law still performs best (by a wider margin) while the LeCun–normalized variant consistently surpasses the baseline. The oscillatory patterns observed are due to the fixed learning rate, but crucially, the fluctuations occur around equilibrium points significantly lower than the convergence levels of the competing schemes.

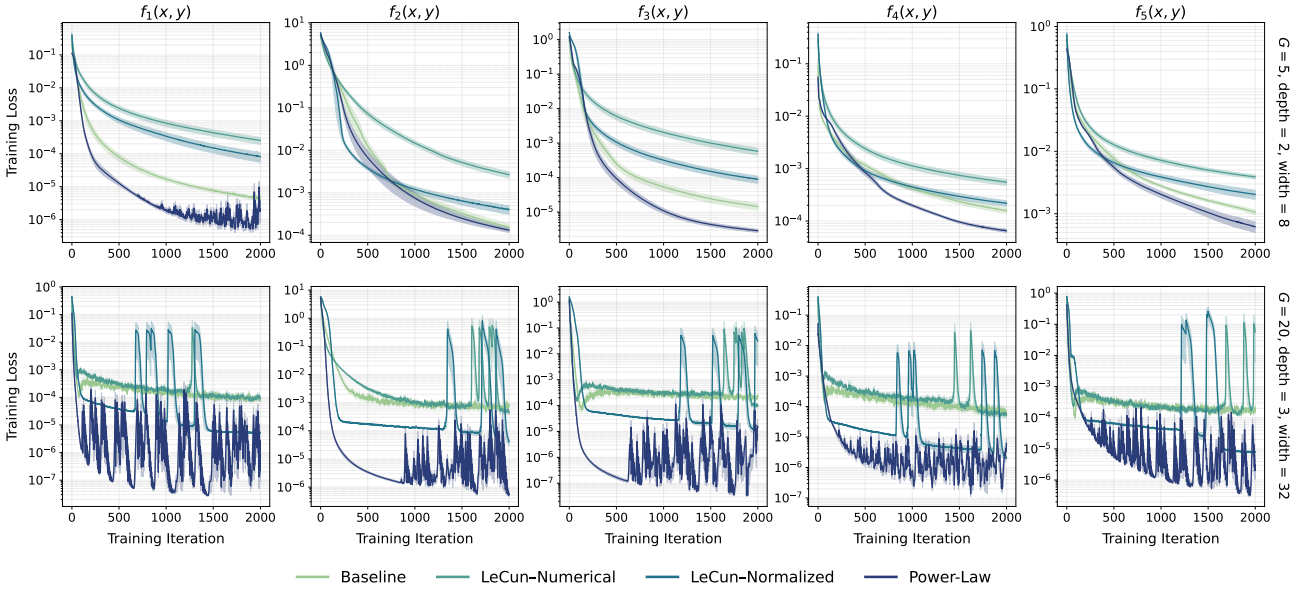

Figure 1. Training loss curves (with shaded standard error) for small and large architectures across all four initialization strategies. For the small architecture, the power-law initialization performs best, with the baseline outperforming both LeCun-based schemes. In the large architecture setting, the power-law remains dominant, while the LeCun–normalized strategy consistently surpasses the baseline.

5. Conclusion

We investigated initialization strategies for KANs, focusing on variance-preserving and empirical power-law schemes. Through experiments on function fitting tasks, we showed that suitable initialization choices can lead to substantial performance improvements over the baseline, especially for deeper or wider architectures. Our findings suggest that even small-scale experimentation can reveal important insight in the dynamics of KAN training.

5.1. Limitations and Future Work

Of course, this study focuses solely on function fitting tasks; evaluating initialization strategies on more complex and resource-intensive problems, such as PDEs or image-based benchmarks, remains an open direction. Additionally, we restricted our analysis to KANs with spline basis functions, though alternative basis representations are also unexplored. Finally, our variance-based initializations do not account for gradient propagation; future work could investigate Glorot-like

schemes that explicitly consider both forward and backward signal flow. Addressing these limitations is essential for building a more complete understanding of initialization in KANs, and we intend to explore these directions in follow-up studies.

Data Availability

The data corresponding to the full grid-search experiments described in Section 3 as well as the code to reproduce the experimental results of Section 4 can be found as Supplementary Material in the GitHub repository for MOSS, ICML 2025, under `submissions/submission-8`.

References

Abueidda, D. W., Pantidis, P., and Mobasher, M. E. DeepOKAN: Deep operator network based on Kolmogorov Arnold networks for mechanics problems. *Comput. Methods Appl. Mech. Eng.*, 436:117699, 2025. doi: https://doi.org/10.1016/j.cma.2024.117699. URL https://www.sciencedirect.com/science/article/pii/S0045782524009538.

Alter, T., Lapid, R., and Sipper, M. On the robustness of Kolmogorov–Arnold networks: An adversarial perspective. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=uafxqhImPM.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pp. 249–256, 2010. URL https://proceedings.mlr.press/v9/glorot10a.html.

Guilhoto, L. F. and Perdikaris, P. Deep learning alternatives of the Kolmogorov superposition theorem. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=SyVPiehSbg.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

Howard, A. A., Jacob, B., Murphy, S. H., Heinlein, A., and Stinis, P. Finite basis Kolmogorov–Arnold networks: domain decomposition for data-driven and physics-informed problems, 2024. URL https://arxiv.org/abs/2406.19662.

Huang, X. S., Perez, F., Ba, J., and Volkovs, M. Improving transformer optimization through better initialization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pp. 4475–4483, 2020. URL https://proceedings.mlr.press/v119/huang20f.html.

Ioffe, S. and Szegedy, C. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, pp. 448–456, 2015.

Kashefi, A. Kolmogorov–Arnold PointNet: Deep learning for prediction of fluid fields on irregular geometries. *Comput. Methods Appl. Mech. Eng.*, 439:117888, 2025. doi: https://doi.org/10.1016/j.cma.2025.117888. URL https://www.sciencedirect.com/science/article/pii/S0045782525001604.

Kolmogorov, A. K. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373, 1957.

Kundu, A., Sarkar, A., and Sadhu, A. KANQAS: Kolmogorov–Arnold network for quantum architecture search. *EPJ Quantum Technol.*, 11:76, 2024. doi: https://doi.org/10.1140/epjqt/s40507-024-00289-z.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In Orr, G. B. and Müller, K.-R. (eds.), *Neural Networks: Tricks of the Trade*, pp. 9–50. Springer, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2.

Liu, Z., Ma, P., Wang, Y., Matusik, W., and Tegmark, M. Kan 2.0: Kolmogorov–Arnold networks meet science, 2024. URL https://arxiv.org/abs/2408.10205.

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljacic, M., Hou, T. Y., and Tegmark, M. KAN: Kolmogorov–Arnold networks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Ozo7qJ5vZi.

Mishkin, D. and Matas, J. All you need is a good init. In *4th International Conference on Learning Representations, ICLR 2016*, 2016. URL https://arxiv.org/abs/1511.06422.

Poeta, E., Giobergia, F., Pastor, E., Cerquitelli, T., and Baralis, E. A benchmarking study of Kolmogorov–Arnold networks on tabular data. In *2024 IEEE 18th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 1–6, 2024. doi: 10.1109/AICT61888.2024.10740444.

Rigas, S. and Papachristou, M. jaxKAN: A unified JAX framework for Kolmogorov–Arnold networks. *Journal of Open Source Software*, 10(108):7830, 2025. doi: 10.21105/joss.07830. URL https://doi.org/10.21105/joss.07830.

Rigas, S., Papachristou, M., Papadopoulos, T., Anagnostopoulos, F., and Alexandridis, G. Adaptive training of grid-dependent physics-informed Kolmogorov–Arnold networks. *IEEE Access*, 12:176982–176998, 2024. doi: 10.1109/ACCESS.2024.3504962.

Shukla, K., Toscano, J. D., Wang, Z., Zou, Z., and Karniadakis, G. E. A comprehensive and FAIR comparison between MLP and KAN representations for differential equations and operator networks. *Comput. Methods Appl. Mech. Eng.*, 431:117290, 2024. doi: https://doi.org/10.1016/j.cma.2024.117290. URL https://www.sciencedirect.com/science/article/pii/S0045782524005462.

Skorski, M., Temperoni, A., and Theobald, M. Revisiting weight initialization of deep neural networks. In *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157, pp. 1192–1207, 2021. URL https://proceedings.mlr.press/v157/skorski21a.html.

Wang, Y., Siegel, J. W., Liu, Z., and Hou, T. Y. On the expressiveness and spectral bias of KANs. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=ydlDRUuGm9.

Wang, Y., Sun, J., Bai, J., Anitescu, C., Eshaghi, M. S., Zhuang, X., Rabczuk, T., and Liu, Y. Kolmogorov–Arnold-informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on Kolmogorov–Arnold networks. *Comput. Methods Appl. Mech. Eng.*, 433:117518, 2025b. doi: https://doi.org/10.1016/j.cma.2024.117518. URL https://www.sciencedirect.com/science/article/pii/S0045782524007722.

Yu, R., Yu, W., and Wang, X. KAN or MLP: A fairer comparison, 2024. URL https://arxiv.org/abs/2407.16674.

Zhang, X. and Zhou, H. Generalization bounds and model complexity for Kolmogorov–Arnold networks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=q5zMyAUhGx.

A. Variance-Preserving Initialization

In this appendix, we provide a derivation of Eqs. (2) from the main text and obtain more explicit expressions under the assumption that network inputs are independently sampled from a uniform distribution $\mathcal{U}(-1,1)$. Assuming statistical independence between each term in the outer sum of Eq. (1) and requiring the output variance to match the input variance, one finds

$$\mathrm{Var}(x_i) = n_{\text{in}}\mathrm{Var}\left[r_{ji}\cdot R(x_i) + c_{ji}\sum_{m=1}^{G+k} b_{jim}\cdot B_m(x_i)\right], \tag{5}$$

where the right-hand side contains the variance of a sum of $G+k+1$ terms: one residual term and $G+k$ spline basis terms. Since these terms are not mutually uncorrelated, we adopt a simplifying assumption that the total variance is approximately equipartitioned across all components,[2] allowing us to bypass pairwise covariance terms. This leads to the following expressions for the residual and spline basis terms, respectively:

$$\frac{\mathrm{Var}(x_i)}{G+k+1} = n_{\text{in}}\mathrm{Var}\left[r_{ji}\cdot R(x_i)\right], \qquad \frac{\mathrm{Var}(x_i)}{G+k+1} = n_{\text{in}}\mathrm{Var}\left[b_{jim}\cdot B_m(x_i)\right]. \tag{6}$$

Since the trainable weights r_{ji} are independent of the residual function $R(x_i)$, the variance of their product becomes

$$\begin{aligned}\mathrm{Var}\left[r_{ji}\cdot R(x_i)\right] &= \underbrace{\mathbb{E}^2(r_{ji})}_{=0}\mathrm{Var}\left[R(x_i)\right] + \mathbb{E}^2\left[R(x_i)\right]\mathrm{Var}(r_{ji}) + \mathrm{Var}(r_{ji})\,\mathrm{Var}\left[R(x_i)\right] \\ &= \mathrm{Var}(r_{ji})\left\{\mathrm{Var}\left[R(x_i)\right] + \mathbb{E}^2\left[R(x_i)\right]\right\} = \sigma_r^2\,\mathbb{E}\left[R^2(x_i)\right]\end{aligned} \tag{7}$$

and, in a completely analogous manner, we find

$$\mathrm{Var}\left[b_{jim}\cdot B_m(x_i)\right] = \sigma_b^2\,\mathbb{E}\left[B_m^2(x_i)\right]. \tag{8}$$

Substitution of the expressions of Eqs. (7), (8) into Eqs. (6) yields Eq. (2) from Section 3.1. If we further assume that $x_i \sim \mathcal{U}(-1,1)$, so that $\mathrm{Var}(x_i) = 1/3$, and that the residual function is the SiLU, we can explicitly compute the expected squared residual function as:

$$\begin{aligned}\mathbb{E}\left[R^2(x_i)\right] &= \frac{1}{2}\int_{-1}^{1}\frac{x^2}{\left(1+e^{-x}\right)^2}\,dx \\ &= \left\{(x+1)\,\mathrm{Li}_2\left(-e^x\right) - \mathrm{Li}_3\left(-e^x\right) + \frac{1}{2}x\left[(x+2)\ln\left(1+e^x\right) - \frac{x\cdot e^x}{e^x+1}\right]\right\}_{-1}^{1} \approx 9.45\cdot 10^{-2},\end{aligned} \tag{9}$$

where

$$\mathrm{Li}_s(z) = \sum_{k=1}^{\infty}\frac{z^k}{k^s}. \tag{10}$$

This suggests initializing r_{ji} from a distribution with zero mean and standard deviation

$$\sigma_r \approx \frac{1.878}{\sqrt{n_{\text{in}}(G+k+1)}}. \tag{11}$$

[2] This assumption does not necessarily hold in general. For example, one could consider a 50%–50% split between the residual and basis function terms. We experimented with this alternative and found that it yielded poorer results compared to the variance partitioning that leads to Eqs. (2).

An equivalent closed-form expression for σ_b is not available, as the B_m spline basis functions are inherently grid-dependent. This motivates the two "LeCun-like" initializations presented in Section 3: the LeCun–numerical strategy computes $\mathbb{E}\left[B_m^2(x_i)\right]$ numerically for a given grid, while the LeCun–normalized one uses the normalized basis functions of Eq. (3), for which $\mathbb{E}\left[B_m^2(x_i)\right] = 1$, yielding $\sigma_b = \left[3n_{\text{in}}(G+k+1)\right]^{-1/2}$.

B. Implementation Details

For the purposes of this work, we study five two-dimensional functions ranging from simple expressions to more complex, nonlinear, or piecewise-defined forms. Specifically, we consider the following functions in the $[-1,1]\times[-1,1]$ domain:

- $f_1(x,y) = xy$
- $f_2(x,y) = \exp\left(\sin(\pi x) + y^2\right)$
- $f_3(x,y) = I_1(x) + \exp\left[\exp(-|y|)\, I_1(y)\right] + \sin(xy)$
- $f_4(x,y) = S\Big[f_3(x,y) + \text{erf}^{-1}(y)\Big] \times C\Big[f_3(x,y) + \text{erf}^{-1}(y)\Big]$
- $f_5(x,y) = y \cdot \text{sgn}(0.5 - x) + \text{erf}(x) \cdot \min\left(xy, \frac{1}{xy}\right)$

where $I_1(x)$ is the modified Bessel function of first order, $\text{sgn}(x)$ is the sign function, $\text{erf}(x)$ is the error function and $S(x)$, $C(x)$ are the Fresnel integral functions defined as

$$S(x) = \int_0^x \sin\left(\frac{\pi t^2}{2}\right) dt, \qquad C(x) = \int_0^x \cos\left(\frac{\pi t^2}{2}\right) dt. \tag{12}$$

The reference surfaces for these functions are shown in Figure 2.

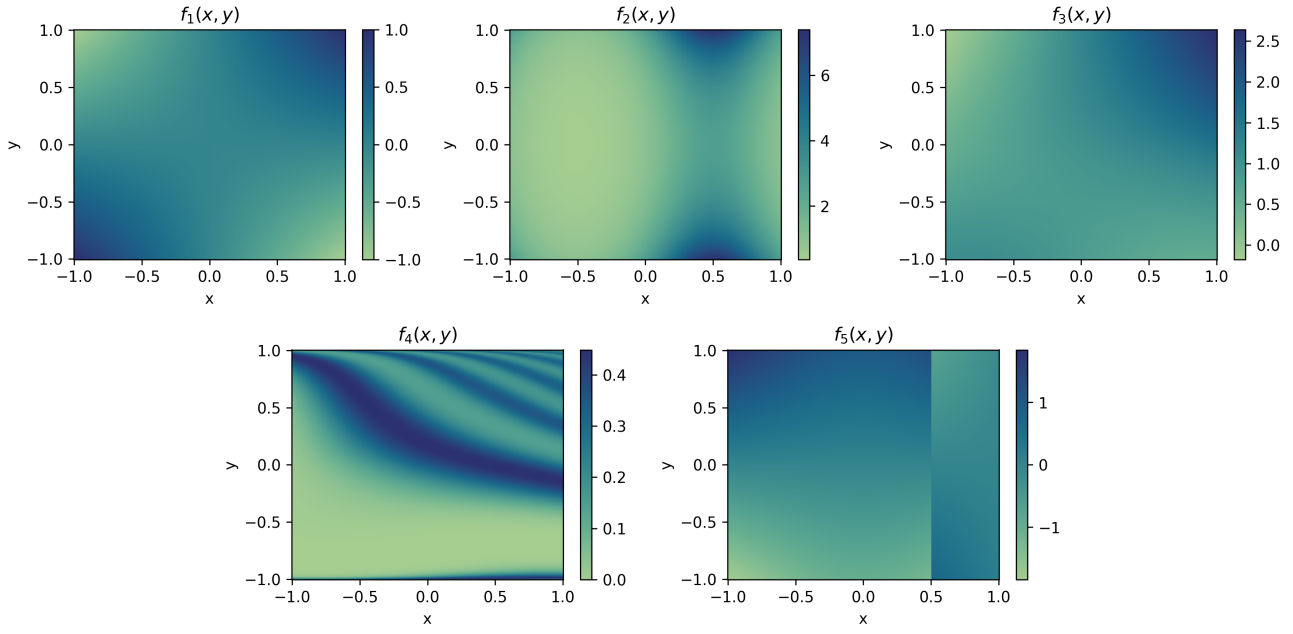

Figure 2. Reference surfaces for the five two-dimensional target functions f_1 through f_5 used in the function fitting experiments.

The KAN models used to fit these functions utilize spline basis functions of order $k = 3$, defined over an augmented, uniform grid within the $[-1,1]$ domain (Liu et al., 2025). Training is performed using the Adam optimizer with a fixed

learning rate of 10^{-3}, with the objective of minimizing the mean squared error between the predicted and reference function values. For each target function $f_i(x, y)$, with $i = 1, \ldots, 5$, we generate 4000 random input samples uniformly distributed over the domain $[-1, 1] \times [-1, 1]$, and evaluate the corresponding outputs to serve as ground truth during training.

C. Supplementary Experimental Results

This appendix contains additional experimental results that expand on the findings summarized in Section 4 of the main text. Figures 3 – 6 present the full set of experimental results summarized in Table 1 of the main text. These heatmaps show, for each combination of function and architecture, the performance of the four initialization strategies. Each figure corresponds to a different value of G ($G = 5$, 10, 20, and 40 respectively), with columns representing initialization strategies and rows representing target functions. For the power-law scheme, only the (α, β) configuration that achieves the lowest training loss for each architecture is shown.

To provide a finer-grained view into the performance landscape of the power-law initialization, Figures 7 – 10 present a detailed grid search over (α, β) configurations for the function $f_3(x, y)$ across the same four grid sizes. In these heatmaps, the horizontal axis corresponds to α and the vertical axis to β, while rows and columns represent different architectural widths and depths, respectively. These results help justify the selection of $\alpha = 0.25$ and $\beta = 1.75$ in the case study comparison shown in Figure 1 of the main text.

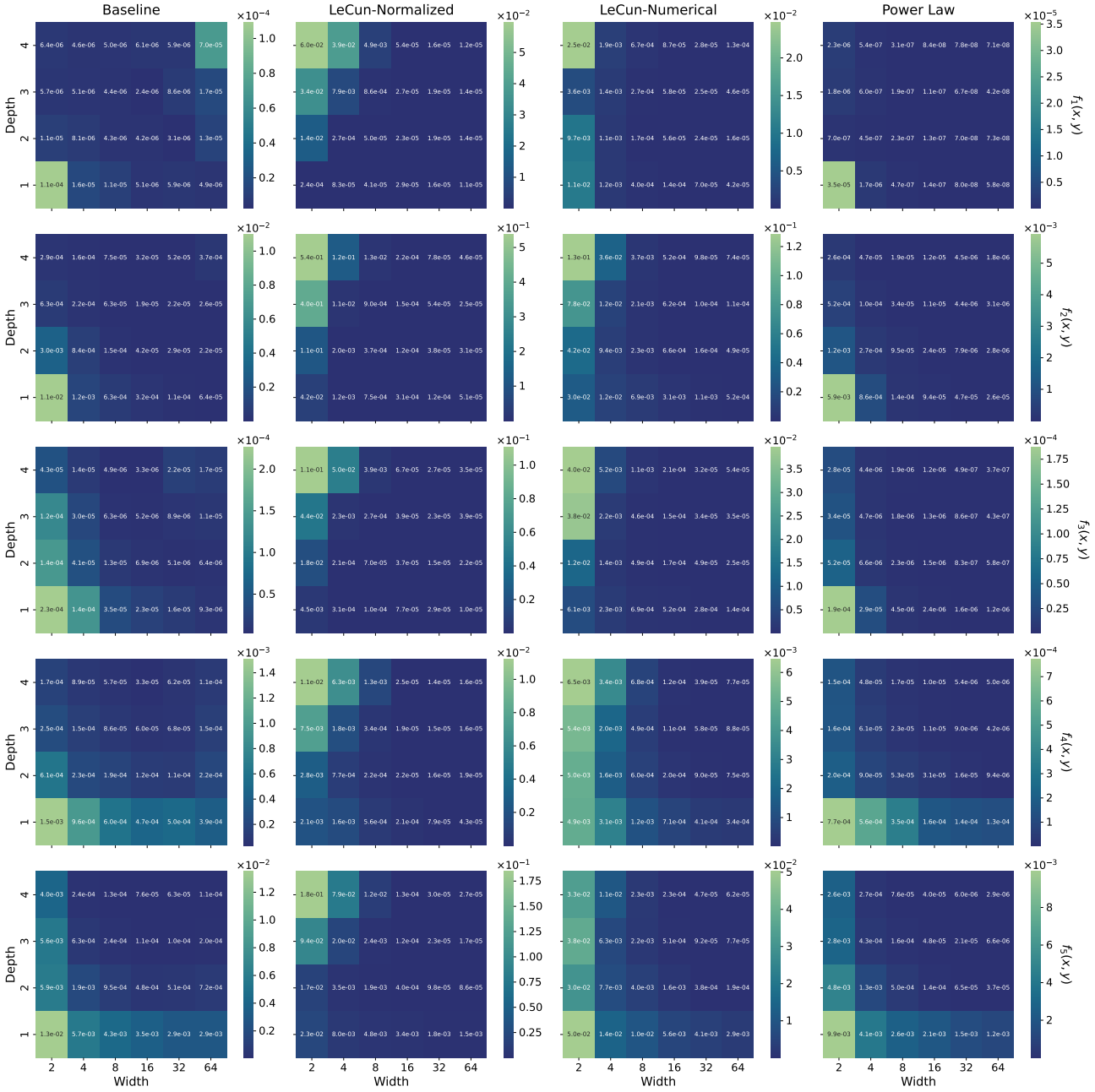

Figure 3. Training loss heatmaps for all four initialization strategies (Baseline, LeCun–Normalized, LeCun–Numerical, and Power-Law) on each of the five target functions, with grid size $G = 5$. Each row corresponds to a different function, and each column to an initialization method. Within each heatmap, the horizontal axis represents the hidden layer width, the vertical axis represents the number of hidden layers and the color indicates the final training loss.

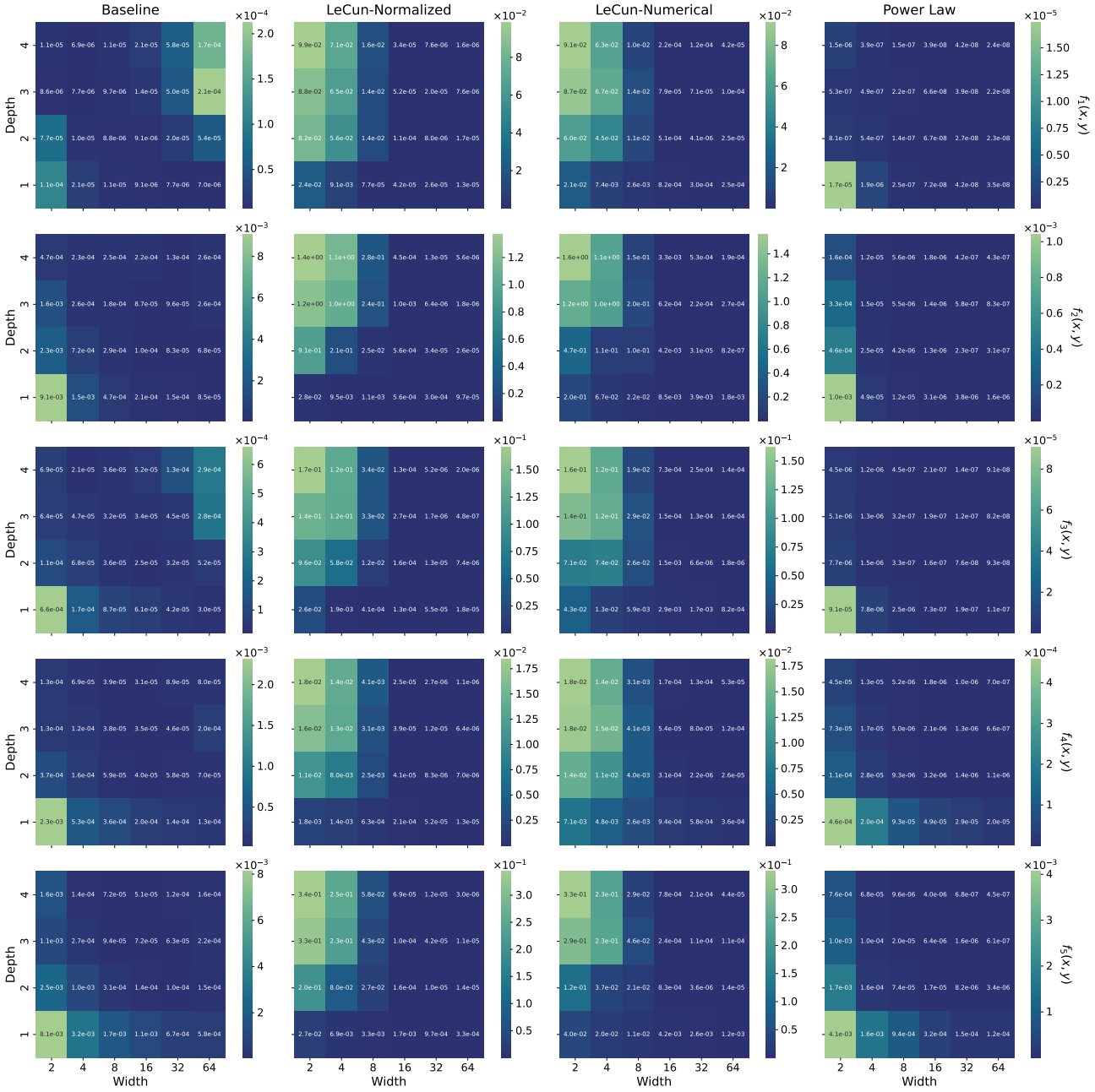

Figure 4. Training loss heatmaps for all four initialization strategies (Baseline, LeCun–Normalized, LeCun–Numerical, and Power-Law) on each of the five target functions, with grid size $G = 10$. Each row corresponds to a different function, and each column to an initialization method. Within each heatmap, the horizontal axis represents the hidden layer width, the vertical axis represents the number of hidden layers and the color indicates the final training loss.

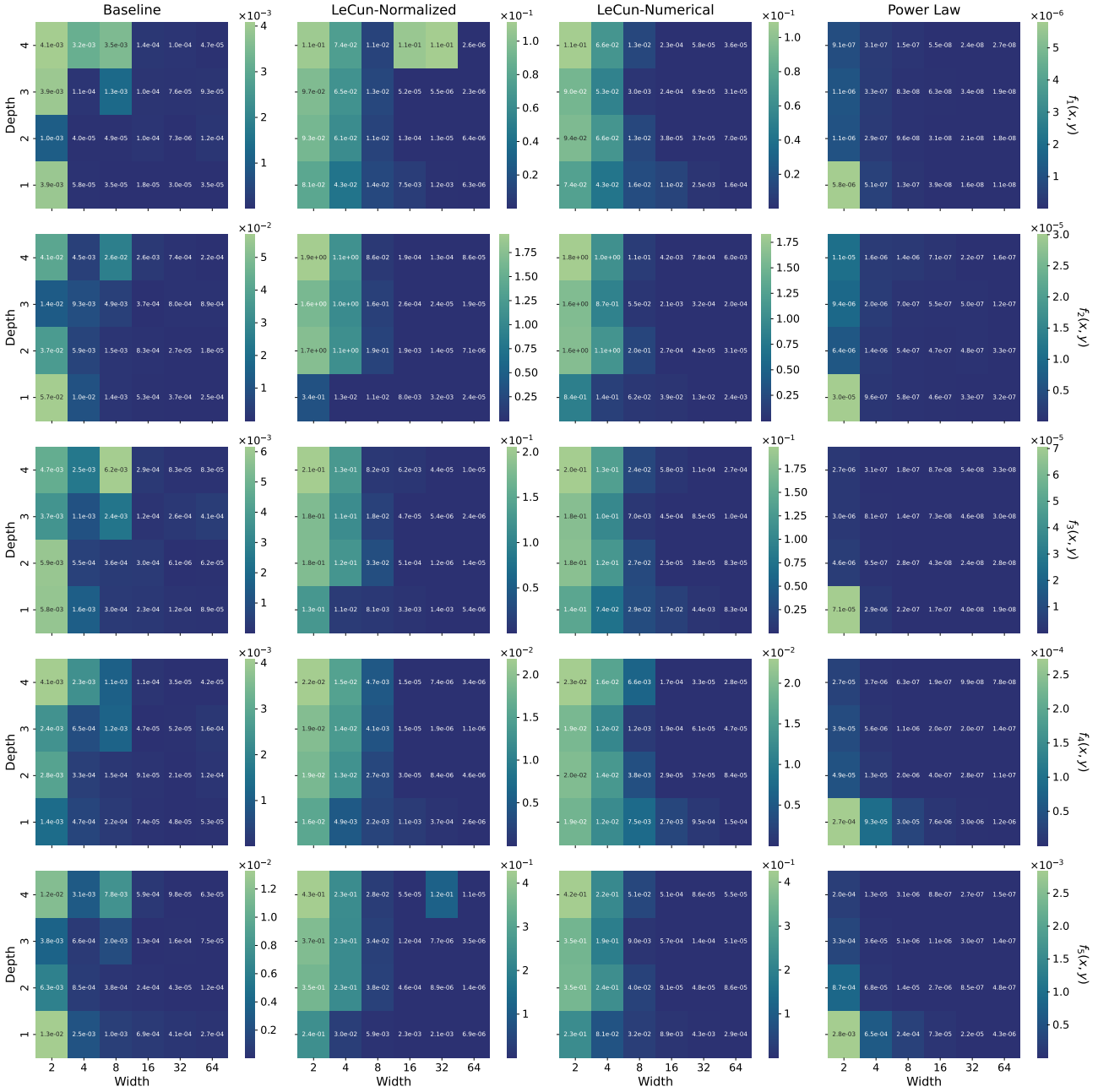

Figure 5. Training loss heatmaps for all four initialization strategies (Baseline, LeCun–Normalized, LeCun–Numerical, and Power-Law) on each of the five target functions, with grid size $G = 20$. Each row corresponds to a different function, and each column to an initialization method. Within each heatmap, the horizontal axis represents the hidden layer width, the vertical axis represents the number of hidden layers and the color indicates the final training loss.

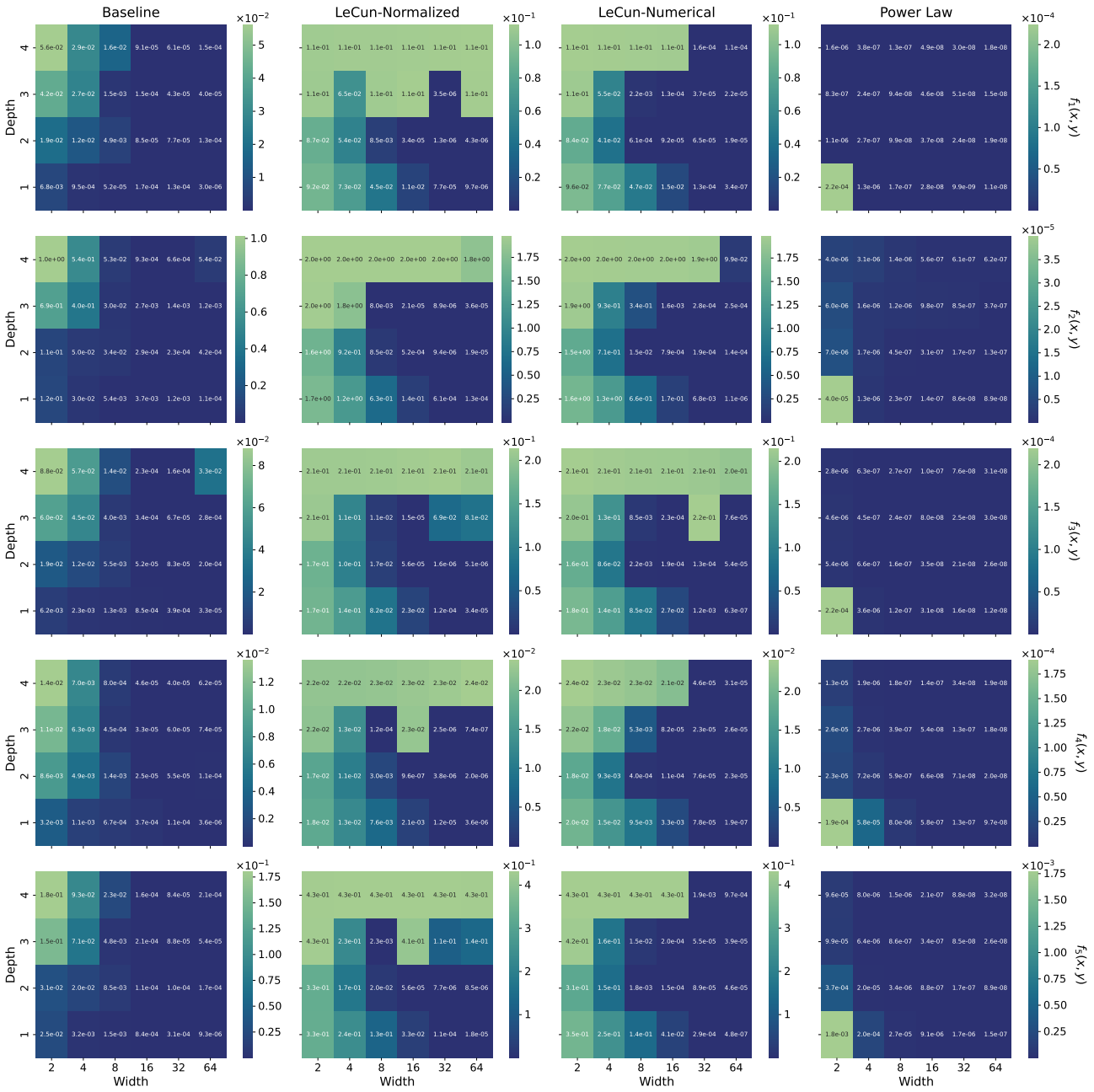

Figure 6. Training loss heatmaps for all four initialization strategies (Baseline, LeCun–Normalized, LeCun–Numerical, and Power-Law) on each of the five target functions, with grid size $G = 40$. Each row corresponds to a different function, and each column to an initialization method. Within each heatmap, the horizontal axis represents the hidden layer width, the vertical axis represents the number of hidden layers and the color indicates the final training loss.

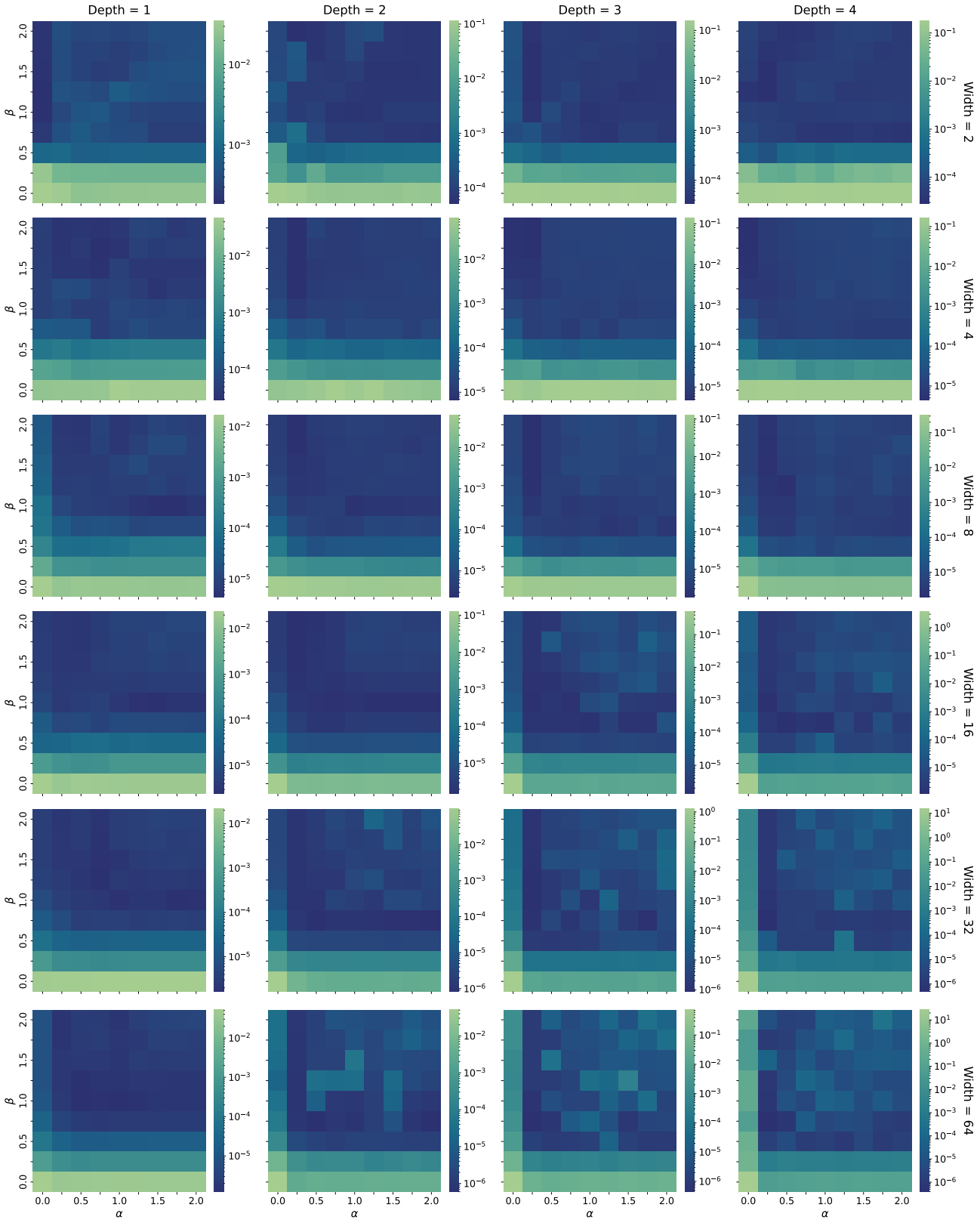

Figure 7. Grid search for the power-law initialization applied to function $f_3\,(x, y)$ for $G = 5$. Each heatmap corresponds to an architecture, with the horizontal and vertical axis representing α and β, respectively, and color denoting final training loss.

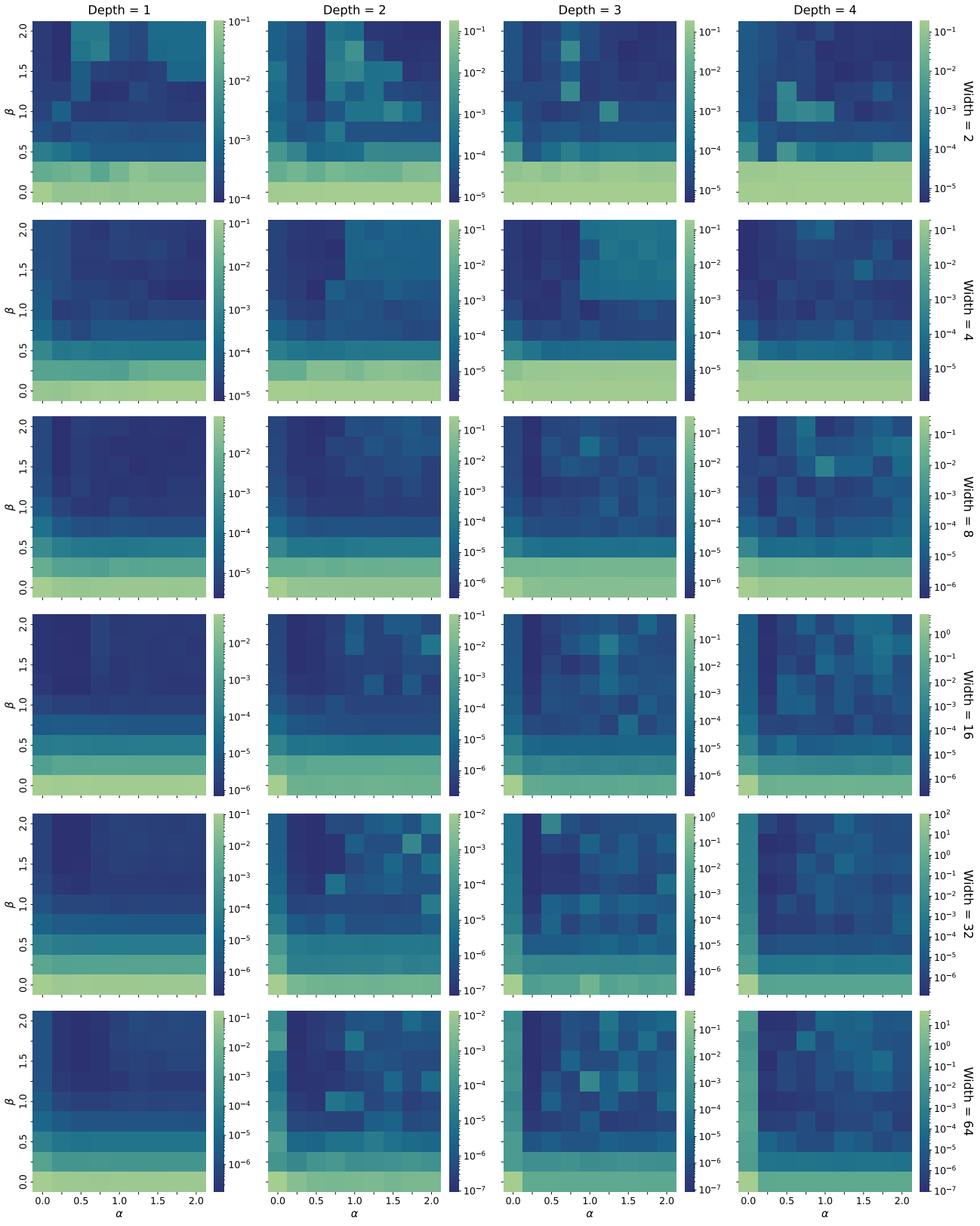

Figure 8. Grid search for the power-law initialization applied to function $f_3\,(x, y)$ for $G = 10$. Each heatmap corresponds to an architecture, with the horizontal and vertical axis representing α and β, respectively, and color denoting final training loss.

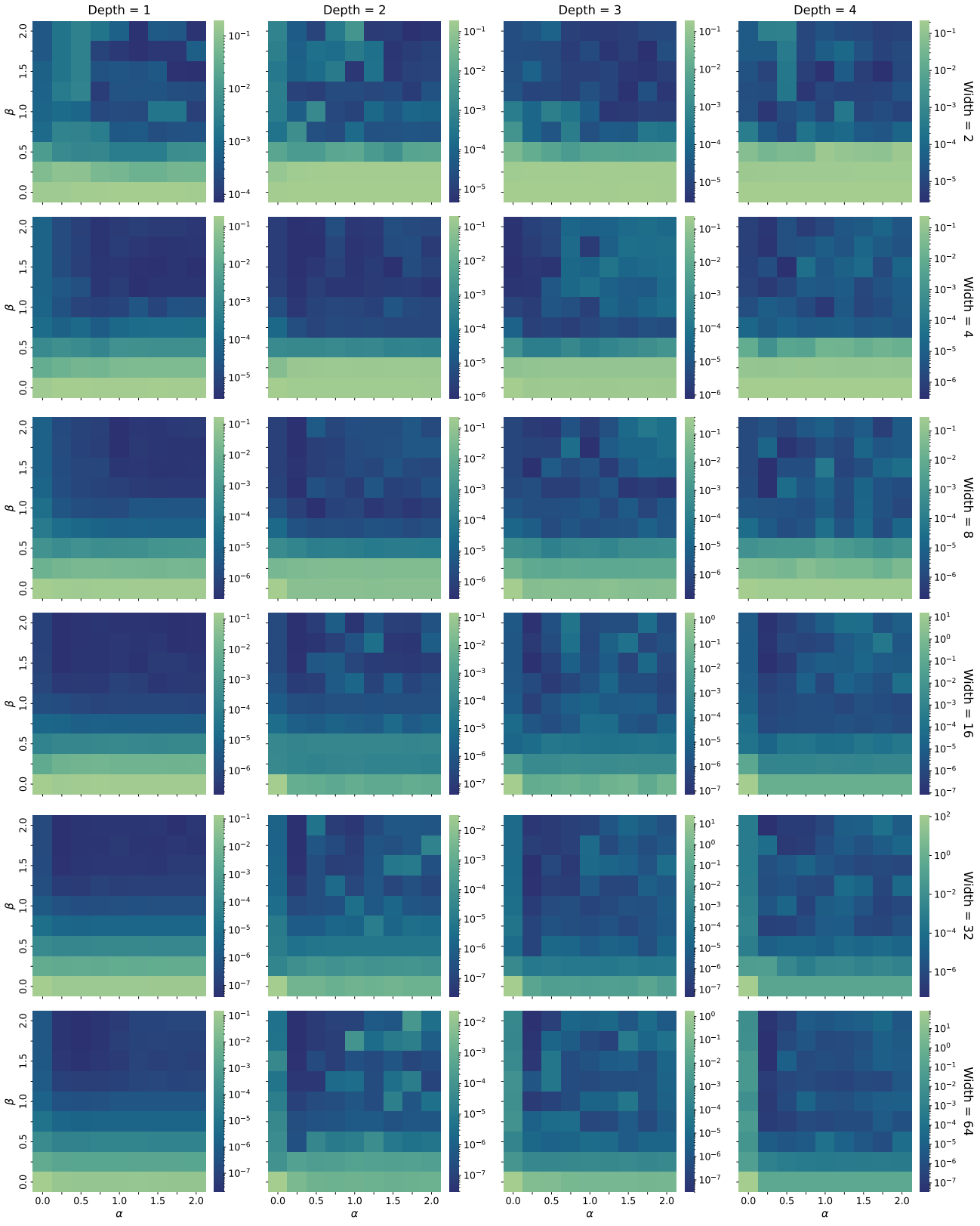

Figure 9. Grid search for the power-law initialization applied to function $f_3\,(x, y)$ for $G = 20$. Each heatmap corresponds to an architecture, with the horizontal and vertical axis representing α and β, respectively, and color denoting final training loss.

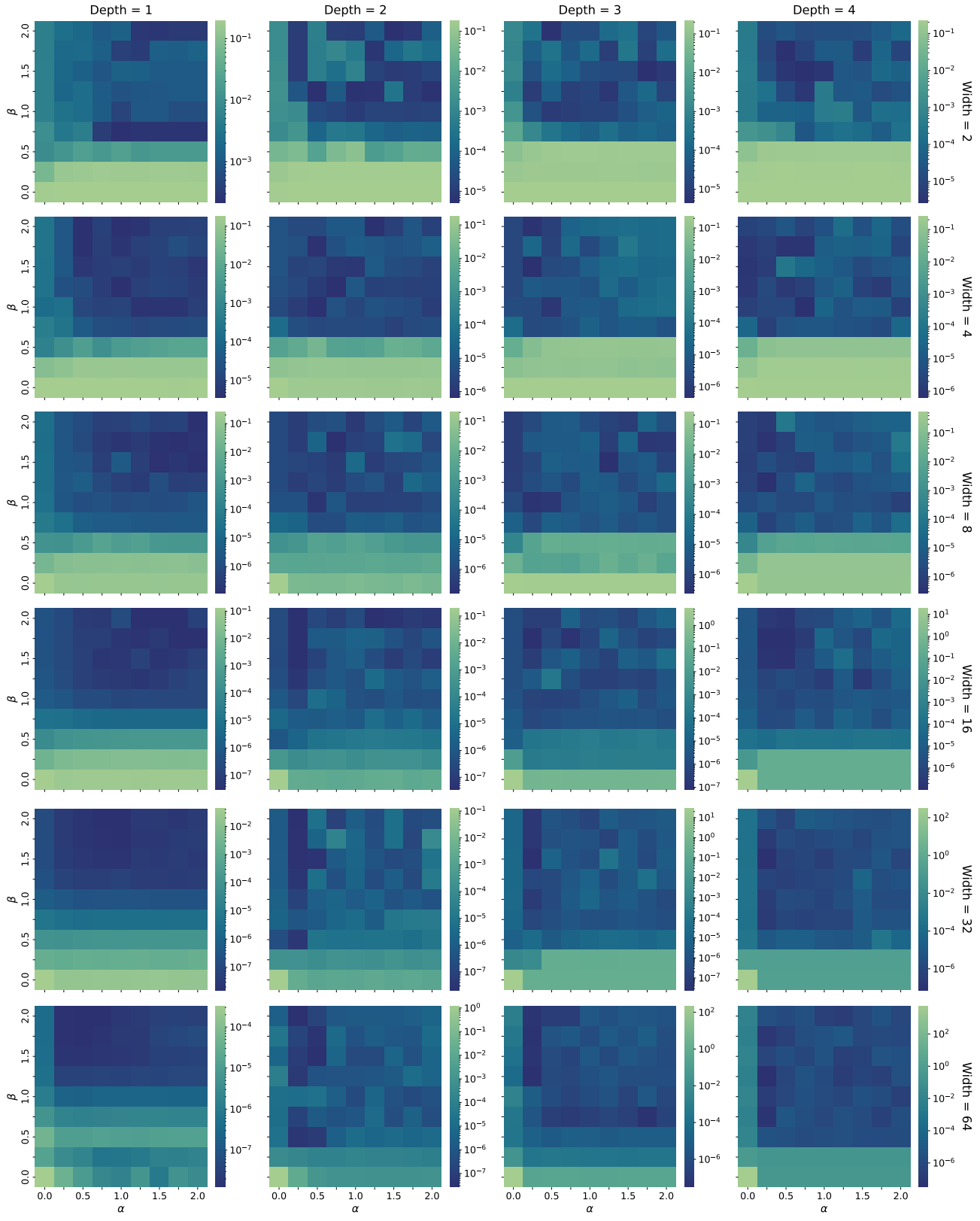

Figure 10. Grid search for the power-law initialization applied to function $f_3(x, y)$ for $G = 40$. Each heatmap corresponds to an architecture, with the horizontal and vertical axis representing α and β, respectively, and color denoting final training loss.