
COUNTERFACTUAL TRAINING: TEACHING MODELS PLAUSIBLE AND ACTIONABLE EXPLANATIONS

A PREPRINT

Patrick Altmeyer 

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

p.altmeyer@tudelft.nl

Arie van Deursen

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Cynthia C. S. Liem

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

January 30, 2025

ABSTRACT

Counterfactual Explanations (CE) have emerged as a popular method to explain the predictions made by opaque machine learning models in a post-hoc fashion. We propose a novel approach that leverages counterfactuals during the training phase of models.

Keywords Counterfactual Explanations • Explainable AI

1 Introduction

2 Related Literature

2.1 Background on Counterfactual Explanations

(Wachter, Mittelstadt, and Russell 2017; Joshi et al. 2019; Altmeyer et al. 2024)

2.2 Learning Representations

For example, joint-energy models

2.3 Generalization and Robustness

Sauer and Geiger (2021) generate counterfactual images for MNIST and ImageNet through independent mechanisms (IM): each IM learns class-conditional input distributions over a specific lower-dimensional, semantically meaningful factor, such as *texture*, *shape* and *background*. They demonstrate that using these generated counterfactuals during classifier training improves model robustness. Similarly, Abbasnejad et al. (2020) argue that counterfactuals represent potentially useful training data in machine learning, especially in supervised settings where inputs may be reasonably mapped to multiple outputs. They, too, demonstrate that augmenting the training data of image classifiers can improve generalization.

19 Tenev, Abbasnejad, and Hengel (2020) propose an approach using counterfactuals in training that does not rely on
 20 data augmentation: they argue that counterfactual pairs typically already exist in training datasets. Specifically, their
 21 approach relies on, firstly, identifying similar input samples with different annotations and, secondly, ensuring that the
 22 gradient of the classifier aligns with the vector between pairs of counterfactual inputs using the cosine distance as a loss
 23 function (referred to as *gradient supervision*) (*this might be useful for our task as well*). In the natural language pro-
 24 cessing (NLP) domain, counterfactuals have similarly been used to improve models through data augmentation: Wu et
 25 al. (2021), propose POLYJUICE, a general-purpose counterfactual generator for language models. They demonstrate
 26 empirically that augmenting training data through POLYJUICE counterfactuals improves robustness in a number of
 27 NLP tasks.

28 **2.4 Link to Adversarial Training**

29 Freiesleben (2022) propose two definitional differences between Adversarial Examples (AE) and Counterfactual Ex-
 30 planations (CE): firstly, and more importantly according to the authors, the term AE implies missclassification, which
 31 is not the case for CE (*this might be a useful notion for use to distinguish between adversarials and explanations*
 32 *during training*); secondly, they argue that closeness plays a more critical role in the context of CE but confess that
 33 even counterfactuals that are not close might be relevant explanations. Pawelczyk et al. (2022) show that CE and AE
 34 are equivalent under certain conditions and derive upper bounds on the distances between them.

35 **2.5 Closely Related**

36 Guo, Nguyen, and Yadav (2023) are the first to propose end-to-end training pipeline that includes counterfactual ex-
 37 planations as part of the training procedure. In particular, they propose a specific network architecture that includes
 38 a predictor and CE generator network (*akin a GAN?*), where the parameters of the CE generator network are learn-
 39 able. Counterfactuals are generated during each training iteration and fed back to the predictor network (*here we are*
 40 *aligned*). In contrast, we impose no restrictions on the neural network architecture at all. (*to ensure the one-hot en-*
 41 *coding of categorical features is maintained, they simple use softmax (might be interesting for CE.jl)*) Interestingly,
 42 the authors find that their approach is sensitive to the choice of the loss function: only MSE seems to lead to good
 43 performance. They also demonstrate theoretically, that the objective function is difficult to optimize due to divergent
 44 gradients and suffers from poor adversarial robustness. (*because partial gradients with respect to the classification*
 45 *loss component and the counterfactual validity component point in opposite directions*). To mitigate these issues,
 46 the authors use block-wise gradient descent: they first update with respect to classification loss and then use a second
 47 update with respect to the other loss components (*this might be useful for our task as well*). Ross, Lakkaraju, and
 48 Bastani (2024) propose a way to train models that are guaranteed to provide recourse for individuals with high proba-
 49 bility. The approach builds on adversarial training (*here we are aligned*), where in this context adversarial examples
 50 are actively encouraged to exist, but only target attacks with respect to the positive class. The proposed method allows
 51 for imposing a set of actionable recourse ex-ante: for example, users can impose mutability constraints for features
 52 (*here we are aligned*). (*To solve their objective function more efficiently, they use a first-order Taylor approximation*
 53 *to approximate the recourse loss component (might be applicable in our case)*)

54 Luu and Inoue (2023) introduce Counterfactual Adversarial Training (CAT) with intention of improving generalization
 55 and robustness of language models. Specifically, they propose to proceed as follows: firstly, identify training samples
 56 that are subject to high predictive uncertainty (entropy); secondly, generate counterfactual explanations for those
 57 samples; and, finally, finetune the model on the augmented dataset that includes the generated counterfactuals.

58 **3 Counterfactual Training**

59 **4 Experiments**

60 **4.1 Experimental Setup**

61 **4.2 Experimental Results**

62 **5 Discussion**

63 **6 Conclusion**

64 **References**

65 Abbasnejad, Ehsan, Damien Teney, Amin Parvaneh, Javen Shi, and Anton van den Hengel. 2020. “Counterfactual
 66 Vision and Language Learning.” In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*
 67 (*CVPR*), 10041–51. <https://doi.org/10.1109/CVPR42600.2020.01006>.

- 68 Altmeyer, Patrick, Mojtaba Farmanbar, Arie van Deursen, and Cynthia CS Liem. 2024. “Faithful Model Explanations
 69 Through Energy-Constrained Conformal Counterfactuals.” In *Proceedings of the AAAI Conference on Artificial*
 70 *Intelligence*, 38:10829–37. 10.
- 71 Freiesleben, Timo. 2022. “The Intriguing Relation Between Counterfactual Explanations and Adversarial Examples.”
 72 *Minds and Machines* 32 (1): 77–109.
- 73 Guo, Hangzhi, Thanh H. Nguyen, and Amulya Yadav. 2023. “CounterNet: End-to-End Training of Prediction Aware
 74 Counterfactual Explanations.” In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery*
 75 *and Data Mining*, 577–89. KDD ’23. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/3580305.3599290>.
- 76 Joshi, Shalmali, Oluwasanmi Koyejo, Warut Vigitbenjaronk, Been Kim, and Joydeep Ghosh. 2019. “Towards Realistic
 77 Individual Recourse and Actionable Explanations in Black-Box Decision Making Systems.” <https://arxiv.org/abs/1907.09615>.
- 78 Luu, Hoai Linh, and Naoya Inoue. 2023. “Counterfactual Adversarial Training for Improving Robustness of Pre-
 79 Trained Language Models.” In *Proceedings of the 37th Pacific Asia Conference on Language, Information and*
 80 *Computation*, 881–88.
- 81 Pawelczyk, Martin, Chirag Agarwal, Shalmali Joshi, Sohini Upadhyay, and Himabindu Lakkaraju. 2022. “Exploring
 82 Counterfactual Explanations Through the Lens of Adversarial Examples: A Theoretical and Empirical Analysis.”
 83 In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, edited by Gustau
 84 Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, 151:4574–94. Proceedings of Machine Learning Research.
 85 PMLR. <https://proceedings.mlr.press/v151/pawelczyk22a.html>.
- 86 Ross, Alexis, Himabindu Lakkaraju, and Osbert Bastani. 2024. “Learning Models for Actionable Recourse.” In
 87 *Proceedings of the 35th International Conference on Neural Information Processing Systems*. NIPS ’21. Red
 88 Hook, NY, USA: Curran Associates Inc.
- 89 Sauer, Axel, and Andreas Geiger. 2021. “Counterfactual Generative Networks.” <https://arxiv.org/abs/2101.06046>.
- 90 Teney, Damien, Ehsan Abbasnedjad, and Anton van den Hengel. 2020. “Learning What Makes a Difference from
 91 Counterfactual Examples and Gradient Supervision.” In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part x 16*, 580–99. Springer.
- 92 Wachter, Sandra, Brent Mittelstadt, and Chris Russell. 2017. “Counterfactual Explanations Without Opening the Black
 93 Box: Automated Decisions and the GDPR.” *Harv. JL & Tech.* 31: 841. <https://doi.org/10.2139/ssrn.3063289>.
- 94 Wu, Tongshuang, Marco Tulio Ribeiro, Jeffrey Heer, and Daniel Weld. 2021. “Polyjuice: Generating Counterfactuals
 95 for Explaining, Evaluating, and Improving Models.” In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, edited by Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, 6707–23. Online:
 96 Association for Computational Linguistics. <https://doi.org/10.18653/v1/2021.acl-long.523>.

102 **A Training Details**

103 **A.1 Initial Grid Search**

104 For the initial round of experiments we

105 **A.1.1 Generator Parameters**

106 The hyperparameter grids for the first investigation of the effect of generator parameters are shown in Parameters A.1
107 and Parameters A.2.

108 **Parameters A.1 (Training Phase).**

- 109 • Generator Parameters:
 - 110 – λ_{cost} : 0.0, 0.001, 0.1
 - 111 – λ_{div} : 0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10.0, 15.0
 - 112 – Learning Rate: 1.0
 - 113 – Maximum Iterations: 20, 50, 100
 - 114 – Optimizerimizer: sgd
- 115 • Generator: ecco, generic, omni, revise
- 116 • Training Parameters:
 - 117 – Objective: full, vanilla

118 **Parameters A.2 (Evaluation Phase).**

- 119 • Counterfactual Parameters:
 - 120 – Convergence: max_iter
 - 121 – Maximum Iterations: 100
 - 122 – No. Individuals: 100
 - 123 – No. Runs: 5
- 124 • Generator Parameters:
 - 125 – λ_{cost} : 0.0
 - 126 – λ_{div} : 0.1, 0.5, 1.0, 5.0, 10.0, 20.0
 - 127 – Learning Rate: 1.0
 - 128 – Maximum Iterations: 50
 - 129 – Optimizerimizer: sgd

130 **A.1.1.1 Linearly Separable**

- 131 • **Energy Penalty** (Table A1): *ECCo* generally does yield better results than *Vanilla* for higher choices of the
132 energy penalty (10,15) during training. *Generic* performs poorly accross the board. *Omni* seems to have an
133 anchoring effect, in that it never performs terribly but also never as good as the best *ECCo* results. *REVISE*
134 performs poorly across the board.
- 135 • **Cost** (Table A2): Results for all generators (except *Omni*) are quite bad, which can likely be attributed to
136 extremely bad results for some choices of the **Energy Penalty** (results here are averaged). For *ECCo* and
137 *Generic*, higher cost values generally lead to worse results.
- 138 • **Maximum Iterations**: No clear patterns recognizable, so it seems that smaller choices are ok.
- 139 • **Validity**: *ECCo* almost always valid except for very low values during training and high values at evaluation
140 time. *Generic* often has poor validity.
- 141 • **Accuracy**: Seems largely unaffected.

Table A1: Results for Linearly Separable data by energy penalty.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
full	0.01	<i>ECCo</i>	$-9.91 \cdot 10^{11}$	$2.25 \cdot 10^{12}$
full	0.01	<i>Generic</i>	$-5.71 \cdot 10^{17}$	$1.3 \cdot 10^{18}$
full	0.01	Omniscient	-2.54	0.116
full	0.01	<i>REVISE</i>	-15.6	13.2

Continuing table below.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
vanilla	0.01	<i>ECCo</i>	-4.28	3.52
vanilla	0.01	<i>Generic</i>	-4.45	3.47
vanilla	0.01	<i>Omniscient</i>	-5.12	4.46
vanilla	0.01	<i>REVISE</i>	-4.91	4.24
full	0.05	<i>ECCo</i>	$-5.63 \cdot 10^5$	$1.28 \cdot 10^6$
full	0.05	<i>Generic</i>	$-8.35 \cdot 10^{17}$	$1.9 \cdot 10^{18}$
full	0.05	Omniscient	-2.53	0.114
full	0.05	<i>REVISE</i>	-15	12.6
vanilla	0.05	<i>ECCo</i>	-4.4	3.66
vanilla	0.05	<i>Generic</i>	-4.38	3.48
vanilla	0.05	<i>Omniscient</i>	-5.25	4.62
vanilla	0.05	<i>REVISE</i>	-4.94	4.22
full	0.1	<i>ECCo</i>	$-6.74 \cdot 10^5$	$1.53 \cdot 10^6$
full	0.1	<i>Generic</i>	$-1.72 \cdot 10^{11}$	$3.9 \cdot 10^{11}$
full	0.1	Omniscient	-2.56	0.124
full	0.1	<i>REVISE</i>	-15.6	13.2
vanilla	0.1	<i>ECCo</i>	-4.28	3.52
vanilla	0.1	<i>Generic</i>	-4.45	3.48
vanilla	0.1	<i>Omniscient</i>	-5.12	4.46
vanilla	0.1	<i>REVISE</i>	-4.91	4.25
full	0.5	<i>ECCo</i>	-11.8	9.83
full	0.5	<i>Generic</i>	$-1.06 \cdot 10^{18}$	$2.42 \cdot 10^{18}$
full	0.5	Omniscient	-2.54	0.123
full	0.5	<i>REVISE</i>	-15	12.6
vanilla	0.5	<i>ECCo</i>	-4.4	3.65
vanilla	0.5	<i>Generic</i>	-4.38	3.48
vanilla	0.5	<i>Omniscient</i>	-5.25	4.61
vanilla	0.5	<i>REVISE</i>	-4.95	4.22
full	1	<i>ECCo</i>	-11.5	11.1
full	1	<i>Generic</i>	$-1.71 \cdot 10^{11}$	$3.88 \cdot 10^{11}$
full	1	Omniscient	-2.59	0.117
full	1	<i>REVISE</i>	-15.7	13.3
vanilla	1	<i>ECCo</i>	-4.28	3.51
vanilla	1	<i>Generic</i>	-4.44	3.47
vanilla	1	<i>Omniscient</i>	-5.11	4.46
vanilla	1	<i>REVISE</i>	-4.91	4.25
full	5	<i>ECCo</i>	-3.99	3.12
full	5	<i>Generic</i>	$-4.88 \cdot 10^{17}$	$1.11 \cdot 10^{18}$
full	5	Omniscient	-2.53	0.117
full	5	<i>REVISE</i>	-14.6	12.1
vanilla	5	<i>ECCo</i>	-4.4	3.65
vanilla	5	<i>Generic</i>	-4.38	3.48
vanilla	5	<i>Omniscient</i>	-5.25	4.61
vanilla	5	<i>REVISE</i>	-4.95	4.22
full	10	ECCo	-2.31	0.735
full	10	<i>Generic</i>	$-1.7 \cdot 10^{11}$	$3.86 \cdot 10^{11}$
full	10	<i>Omniscient</i>	-2.53	0.117
full	10	<i>REVISE</i>	-15.5	13
vanilla	10	<i>ECCo</i>	-4.28	3.51
vanilla	10	<i>Generic</i>	-4.44	3.47
vanilla	10	<i>Omniscient</i>	-5.12	4.46
vanilla	10	<i>REVISE</i>	-4.91	4.24
full	15	ECCo	-2.01	0.488
full	15	<i>Generic</i>	$-4.91 \cdot 10^{17}$	$1.12 \cdot 10^{18}$
full	15	<i>Omniscient</i>	-2.53	0.116

Continuing table below.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
full	15	<i>REVISE</i>	-14.4	11.7
vanilla	15	<i>ECCo</i>	-4.4	3.65
vanilla	15	<i>Generic</i>	-4.38	3.48
vanilla	15	<i>Omniscient</i>	-5.25	4.6
vanilla	15	<i>REVISE</i>	-4.95	4.23

Table A2: Results for Linearly Separable data by cost penalty.

Objective	$\lambda_{\text{cost}}(\text{train})$	Generator	Value	Std
full	0	<i>ECCo</i>	$-5.32 \cdot 10^3$	$1.21 \cdot 10^4$
full	0	<i>Generic</i>	$-1.03 \cdot 10^{18}$	$2.34 \cdot 10^{18}$
full	0	Omniscient	-2.64	0.125
full	0	<i>REVISE</i>	-15.4	12.9
vanilla	0	<i>ECCo</i>	-4.34	3.58
vanilla	0	<i>Generic</i>	-4.41	3.48
vanilla	0	<i>Omniscient</i>	-5.18	4.54
vanilla	0	<i>REVISE</i>	-4.93	4.23
full	0.001	<i>ECCo</i>	-362	811
full	0.001	<i>Generic</i>	$-2.65 \cdot 10^{17}$	$6.03 \cdot 10^{17}$
full	0.001	Omniscient	-2.49	0.115
full	0.001	<i>REVISE</i>	-15.5	13
vanilla	0.001	<i>ECCo</i>	-4.34	3.58
vanilla	0.001	<i>Generic</i>	-4.41	3.48
vanilla	0.001	<i>Omniscient</i>	-5.18	4.53
vanilla	0.001	<i>REVISE</i>	-4.93	4.23
full	0.1	<i>ECCo</i>	$-3.72 \cdot 10^{11}$	$8.46 \cdot 10^{11}$
full	0.1	<i>Generic</i>	$-4.49 \cdot 10^{14}$	$1.02 \cdot 10^{15}$
full	0.1	Omniscient	-2.5	0.112
full	0.1	<i>REVISE</i>	-14.6	12.2
vanilla	0.1	<i>ECCo</i>	-4.34	3.58
vanilla	0.1	<i>Generic</i>	-4.41	3.48
vanilla	0.1	<i>Omniscient</i>	-5.18	4.54
vanilla	0.1	<i>REVISE</i>	-4.93	4.24

142 A.1.1.2 Moons

- **Energy Penalty** (Table A3): *ECCo* consistently yields better results than *Vanilla*, except for very low choices of the energy penalty during training for which it performs abysmal. *Generic* performs quite badly across the board for high enough choices of the energy penalty at evaluation time. *Omni* has small positive effect. *REVISE* performs poorly across the board.
- **Cost (distance penalty)**: *Generic* generally does better for higher values, while *ECCo* does better for lower values.
- **Maximum Iterations**: No clear patterns recognizable, so it seems that smaller choices are ok.
- **Validity**: *ECCo* generally achieves full validity except for very low choices the energy penalty during training and high choices at evaluation time. *Generic* performs poorly for high choices of the energy penalty during evaluation.
- **Accuracy**: Largely unaffected although *ECCo* suffers a bit for very low choices the energy penalty during training. *REVISE* suffers a lot in general (around 10 percentage points).

Table A3: Results for Moons data by energy penalty.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
full	0.01	<i>ECCo</i>	$-2.8 \cdot 10^{22}$	$6.39 \cdot 10^{22}$
full	0.01	<i>Generic</i>	$-4.89 \cdot 10^{30}$	$1.11 \cdot 10^{31}$
full	0.01	Omniscient	-4.74	5.08
full	0.01	<i>REVISE</i>	-572	$1.25 \cdot 10^3$
vanilla	0.01	<i>ECCo</i>	-15.5	17.3
vanilla	0.01	<i>Generic</i>	-10.9	11.9
vanilla	0.01	<i>Omniscient</i>	-12.7	14.4
vanilla	0.01	<i>REVISE</i>	-11.2	13
full	0.05	<i>ECCo</i>	$-1.55 \cdot 10^{16}$	$3.52 \cdot 10^{16}$
full	0.05	<i>Generic</i>	$-2.22 \cdot 10^{20}$	$5 \cdot 10^{20}$
full	0.05	Omniscient	-4.41	4.48
full	0.05	<i>REVISE</i>	$-1.04 \cdot 10^3$	$2.3 \cdot 10^3$
vanilla	0.05	<i>ECCo</i>	-15.5	17.2
vanilla	0.05	<i>Generic</i>	-11.7	12.8
vanilla	0.05	<i>Omniscient</i>	-12.4	14.1
vanilla	0.05	<i>REVISE</i>	-11.3	13.1
full	0.1	<i>ECCo</i>	$-3.41 \cdot 10^3$	$7.73 \cdot 10^3$
full	0.1	<i>Generic</i>	$-5.22 \cdot 10^{30}$	$1.19 \cdot 10^{31}$
full	0.1	Omniscient	-4.78	5.12
full	0.1	<i>REVISE</i>	-288	594
vanilla	0.1	<i>ECCo</i>	-15.5	17.2
vanilla	0.1	<i>Generic</i>	-10.9	11.9
vanilla	0.1	<i>Omniscient</i>	-12.7	14.4
vanilla	0.1	<i>REVISE</i>	-11.3	13.1
full	0.5	<i>ECCo</i>	-7.09	7.51
full	0.5	<i>Generic</i>	$-1.11 \cdot 10^{31}$	$2.53 \cdot 10^{31}$
full	0.5	Omniscient	-4.58	4.83
full	0.5	<i>REVISE</i>	$-1.19 \cdot 10^3$	$2.64 \cdot 10^3$
vanilla	0.5	<i>ECCo</i>	-15.5	17.2
vanilla	0.5	<i>Generic</i>	-11.7	12.8
vanilla	0.5	<i>Omniscient</i>	-12.4	14.1
vanilla	0.5	<i>REVISE</i>	-11.3	13.1
full	1	<i>ECCo</i>	-6.06	6.33
full	1	<i>Generic</i>	$-1.58 \cdot 10^{33}$	$3.59 \cdot 10^{33}$
full	1	Omniscient	-4.66	4.89
full	1	<i>REVISE</i>	$-1.16 \cdot 10^3$	$2.59 \cdot 10^3$
vanilla	1	<i>ECCo</i>	-15.5	17.3
vanilla	1	<i>Generic</i>	-10.9	11.9
vanilla	1	<i>Omniscient</i>	-12.7	14.4
vanilla	1	<i>REVISE</i>	-11.3	13.1
full	5	ECCo	-2.57	2.07
full	5	<i>Generic</i>	$-1.17 \cdot 10^{28}$	$2.66 \cdot 10^{28}$
full	5	<i>Omniscient</i>	-4.29	4.31
full	5	<i>REVISE</i>	-530	$1.16 \cdot 10^3$
vanilla	5	<i>ECCo</i>	-15.5	17.2
vanilla	5	<i>Generic</i>	-11.7	12.7
vanilla	5	<i>Omniscient</i>	-12.4	14.1
vanilla	5	<i>REVISE</i>	-11.3	13.1
full	10	ECCo	-1.76	0.974
full	10	<i>Generic</i>	$-1.54 \cdot 10^{33}$	$3.51 \cdot 10^{33}$
full	10	<i>Omniscient</i>	-4.44	4.56
full	10	<i>REVISE</i>	$-1.52 \cdot 10^3$	$3.4 \cdot 10^3$
vanilla	10	<i>ECCo</i>	-15.5	17.3

Continuing table below.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
vanilla	10	<i>Generic</i>	-10.9	11.9
vanilla	10	<i>Omniscient</i>	-12.7	14.4
vanilla	10	<i>REVISE</i>	-11.3	13.1
full	15	ECCo	-1.37	0.365
full	15	<i>Generic</i>	$-5.32 \cdot 10^{28}$	$1.21 \cdot 10^{29}$
full	15	<i>Omniscient</i>	-4.34	4.38
full	15	<i>REVISE</i>	-473	$1.03 \cdot 10^3$
vanilla	15	<i>ECCo</i>	-15.5	17.2
vanilla	15	<i>Generic</i>	-11.7	12.8
vanilla	15	<i>Omniscient</i>	-12.4	14.1
vanilla	15	<i>REVISE</i>	-11.3	13.1

155 A.1.1.3 Circles

- 156 • **Energy Penalty** (Table A4): *ECCo* consistently yields better results than *Vanilla*, though primarily for low to
 157 medium choices of the energy penalty ($<=5$) during training. The same goes for *Generic*, which sometimes
 158 outperforms *ECCo* (for small energy penalty at evaluation time). *Omni* does alright for lower energy penalty
 159 at evaluation time, but loses out for higher choices. *REVISE* performs poorly across the board (except very
 160 low choices at evaluation time).
- 161 • **Cost (distance penalty)**: *ECCo* and *Generic* generally achieve the best results when no cost penalty is used
 162 during training. Both *Omni* and *REVISE* are largely unaffected.
- 163 • **Maximum Iterations**: *ECCo* consistently yields better results for higher numbers of iterations. *Generic*
 164 generally does best for a medium number (50). *Omni* is sometimes invalid (???).
- 165 • **Validity**: *ECCo* tends to outperform its *Vanilla* counterpart, though primarily for low to medium choices of
 166 the energy penalty ($<=5$) during training and evaluation. *Vanilla* typically worse across the board.
- 167 • **Accuracy**: Mostly unaffected, but *REVISE* again consistently some deterioration and *ECCo* deteriorates for
 168 high choices of energy penalty during training, reflecting other outcomes above.

Table A4: Results for Circles data by energy penalty.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
full	0.01	ECCo	-1.26	0.423
full	0.01	<i>Generic</i>	-1.49	0.71
full	0.01	<i>Omniscient</i>	-5.21	5.25
full	0.01	<i>REVISE</i>	$-2.71 \cdot 10^{26}$	$6.37 \cdot 10^{26}$
vanilla	0.01	<i>ECCo</i>	-9.33	7.34
vanilla	0.01	<i>Generic</i>	-8.89	6.88
vanilla	0.01	<i>Omniscient</i>	-8.67	6.87
vanilla	0.01	<i>REVISE</i>	-8.65	6.8
full	0.05	<i>ECCo</i>	-1.29	0.397
full	0.05	Generic	-1.21	0.356
full	0.05	<i>Omniscient</i>	-5.08	5.09
full	0.05	<i>REVISE</i>	$-5.91 \cdot 10^{27}$	$1.36 \cdot 10^{28}$
vanilla	0.05	<i>ECCo</i>	-9.35	7.32
vanilla	0.05	<i>Generic</i>	-8.85	6.87
vanilla	0.05	<i>Omniscient</i>	-8.7	6.96
vanilla	0.05	<i>REVISE</i>	-8.52	6.76
full	0.1	ECCo	-1.2	0.383
full	0.1	<i>Generic</i>	-1.5	0.735
full	0.1	<i>Omniscient</i>	-5.17	5.23
full	0.1	<i>REVISE</i>	$-3.06 \cdot 10^{26}$	$7.7 \cdot 10^{26}$
vanilla	0.1	<i>ECCo</i>	-9.33	7.32
vanilla	0.1	<i>Generic</i>	-8.88	6.86
vanilla	0.1	<i>Omniscient</i>	-8.69	6.9

Continuing table below.

Objective	$\lambda_{\text{div}}(\text{train})$	Generator	Value	Std
vanilla	0.1	<i>REVISE</i>	-8.68	6.81
full	0.5	ECCo	-1.12	0.217
full	0.5	<i>Generic</i>	-1.21	0.352
full	0.5	<i>Omniscient</i>	-5.09	5.12
full	0.5	<i>REVISE</i>	$-5.97 \cdot 10^{27}$	$1.37 \cdot 10^{28}$
vanilla	0.5	<i>ECCo</i>	-9.35	7.3
vanilla	0.5	<i>Generic</i>	-8.89	6.92
vanilla	0.5	<i>Omniscient</i>	-8.68	6.93
vanilla	0.5	<i>REVISE</i>	-8.53	6.75
full	1	ECCo	-1.1	0.163
full	1	<i>Generic</i>	-1.49	0.726
full	1	<i>Omniscient</i>	-5.16	5.2
full	1	<i>REVISE</i>	$-3.09 \cdot 10^{26}$	$7.22 \cdot 10^{26}$
vanilla	1	<i>ECCo</i>	-9.34	7.36
vanilla	1	<i>Generic</i>	-8.86	6.85
vanilla	1	<i>Omniscient</i>	-8.7	6.9
vanilla	1	<i>REVISE</i>	-8.69	6.85
full	5	<i>ECCo</i>	-1.75	0.154
full	5	Generic	-1.21	0.363
full	5	<i>Omniscient</i>	-5.14	5.16
full	5	<i>REVISE</i>	$-1.1 \cdot 10^{28}$	$2.5 \cdot 10^{28}$
vanilla	5	<i>ECCo</i>	-9.36	7.32
vanilla	5	<i>Generic</i>	-8.88	6.91
vanilla	5	<i>Omniscient</i>	-8.7	6.93
vanilla	5	<i>REVISE</i>	-8.52	6.73
full	10	<i>ECCo</i>	$-1.02 \cdot 10^6$	$2.32 \cdot 10^6$
full	10	Generic	-1.49	0.702
full	10	<i>Omniscient</i>	-5.13	5.16
full	10	<i>REVISE</i>	$-3.74 \cdot 10^{26}$	$9.09 \cdot 10^{26}$
vanilla	10	<i>ECCo</i>	-9.31	7.33
vanilla	10	<i>Generic</i>	-8.87	6.86
vanilla	10	<i>Omniscient</i>	-8.7	6.89
vanilla	10	<i>REVISE</i>	-8.69	6.83
full	15	<i>ECCo</i>	$-3.31 \cdot 10^{13}$	$7.54 \cdot 10^{13}$
full	15	Generic	-1.22	0.37
full	15	<i>Omniscient</i>	-5.2	5.23
full	15	<i>REVISE</i>	$-9.01 \cdot 10^{27}$	$2.06 \cdot 10^{28}$
vanilla	15	<i>ECCo</i>	-9.38	7.34
vanilla	15	<i>Generic</i>	-8.86	6.87
vanilla	15	<i>Omniscient</i>	-8.69	6.96
vanilla	15	<i>REVISE</i>	-8.51	6.73