Understanding Training Data in Large-Scale Machine Learning

Sang Keun Choe

July 2024
CMU-LTI-24-13

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Eric P. Xing (Chair)	Carnegie Mellon University
Emma Strubell	Carnegie Mellon University
Ruslan Salakhutdinov	Carnegie Mellon University
Kyunghyun Cho	New York University
Roger Grosse	University of Toronto

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Neural Networks, Optimization, Data-Centric AI, Automated Machine Learning, Machine Learning Systems, Machine Learning Tooling

To Betty

Abstract

As the capabilities of large-scale machine learning (ML) systems rapidly improve, reliable development & deployment of these systems is increasingly gaining attention. Based on the premise that ML models are in large part reflections of their training data, this thesis aims to achieve reliable ML by developing *principled*, *scalable*, and *operationalizable* frameworks for understanding the influence of (each) training data on the final ML models. Specifically, in the development phase, such frameworks can be used for a wide range of data curation tasks, including noisy label detection, data pruning, and data reweighting. In the deployment phase, we can enable data attribution/valuation to address newly emerging societal challenges, such as data author compensation and data copyright infringement detection.

Towards this goal, we propose extending the inductive programming framework that likens training data (or inductive biases) in ML to source code in traditional software, noting that source code plays a pivotal role in understanding software. In particular, we design a unit data structure specific to inductive biases in ML, and establish a mathematical structure by projecting each unit onto a gradient space with a local metric defined by the Fisher information matrix (FIM). We show that various mathematical operations on this space, such as inner product and norm, can be used to estimate the influence of each inductive bias on the final behavior of ML models and uncertainty in its predictions, thereby laying the foundation to control and interpret black-box ML models, similar to source code in traditional software. Main challenges in operationalizing the above framework involve scalability, algorithmic instability, and programming interface, due to the inherently high-dimensional and stochastic nature of a gradient space. In this thesis, we specifically choose two tasks of automated data optimization and data influence analyses, respectively for controllability and interpretability, and address these challenges by co-designing ML algorithms, systems, and software. Our systems for both tasks are open-sourced to facilitate research in this direction.

Acknowledgements

First and foremost, I want to express my heartfelt gratitude to my advisor, Eric Xing, whose unwavering support and invaluable guidance have been the bedrock of my doctoral journey. Eric introduced me to the world of machine learning systems, which has undoubtedly been the most significant and positive influence on my research career. Through Eric, I have been fortunate to connect with many outstanding SAILING members, including Han Guo, Caleb Ellington, Hongyi Wang, Mingkai Deng, Willie Neiswanger, Pengtao Xie, Bowen Tan, Aviv Bick, Lingjing Kong, Yonghao Zhuang, Aurick Qiao, Hector Liu, Hao Zhang, Nicholas Ho, Sazan Mahbub, Dacheng Li, Hanlin Zhang, and Hao Zhang.

I owe a debt of gratitude to my late advisor during the MLT years, Jaime Carbonell. While I am deeply saddened that we couldn't continue our research together, his dedication, generosity, and intellect will always be remembered. Jaime remains one of the most respectable individuals I have ever met. Under his guidance, I delved into neural network training dynamics and developed a passion for data research, which became major pillars of my PhD work alongside the machine learning systems research I pursued with Eric.

To my esteemed committee members — Emma Strubell, Ruslan Salakhutdinov, Roger Grosse, and Kyunghyun Cho — I extend my sincere appreciation. Emma's insightful feedback and encouragement during my last two PhD projects were invaluable. It is remarkable that my journey in machine learning research, which began with reading foundational work on neural networks by Kyunghyun and Ruslan, has culminated in their guidance on my thesis. Roger, your mentorship on my final doctoral project — the work I have enjoyed most and am proudest of — has been instrumental. Your support extended beyond academia, proving crucial during my job search. I am excited to continue our collaboration at Anthropic in the coming years.

My collaborators have played a pivotal role in shaping my research. Aurick Qiao kindly helped me initiate my PhD research in machine learning systems by inviting me to join his award-winning Pollux project. Pengtao Xie and Willie Neiswanger closely guided my first two meta-learning projects, teaching me the ropes of research and paper writing. Sanket Vaibhav Mehta, a fellow former student of Jaime Carbonell, has been a valuable sounding board for discussions on neural network training dynamics and meta-learning. Conversations

with Hwijeen Ahn, my distant cousin, have helped me articulate research ideas more clearly and approach my work from new angles. The synergy I found in collaborating with Juhan has been a highlight of my academic journey, and I eagerly anticipate our continued partnership in our new professional roles. I am also grateful to my other collaborators: Kewen Zhao, Minsoo Kang, Youngseog Chung, Adithya Pratapa, and Teruko Mitamura.

To all the friends who have made this journey so memorable, you have been the best part of grad school. Betty's CMU uncles and aunts — Sanket Mehta, Hwijeen Ahn, Jimin Sun, Shaily Bhatt, Lindia Tjuatja, Zora Wang, Clara Na, Jimin Mun, Soyeon Min, Yewon Byun, Adithya Pratapa, Kundan Krishna, Youngseog Chung, and Amanda Bertsch — , you are the reason Betty and I looked forward to coming to the office almost every day. Prakhar Gupta and Kaixin Ma, my former office buddies, thanks for the great times. To my MLT friends, including Yong-Siang Shih, Vikas Raunak, Jay-Yoon Lee, Byungsoo Jeon, and Qingtao Hu, thank you for helping me navigate through the stressful MLT years. I have been incredibly lucky to make so many friends in my apartment building too: Juliana Hougland, Ron Sessa, Lea Salvador, Max Armour, Courtney Miller, Andrew Kuznetsov, Susanna Qiao, Alex Holder, Colton Brown, Taylor Norris, Asher Trockman, Katelyn Morrison, Alex Guseman, April Guseman, Emily Parker, Dan Clayton-Luce, Jack Leech, Melanie Cuddyre, Nam Haslam, and Saci Haslam. You have all helped me improve my English, understand American culture, and feel at home in the US. To my oldest and dearest friends — Yeojoon Youn, Wooje Chang, Phil-Young Lee, Jaeyoon Chun, and Wonbok Lee — thanks for your friendship and always cheering me on.

Lastly, but most importantly, my deepest appreciation goes to my family. Mom, your unconditional love and unwavering support have been my anchor through every challenge and triumph. To my sister, though we may squabble over trivialities, I know with certainty that you are always in my corner when it truly matters. A special mention goes to Betty, my pandemic puppy, who joined me mere months before I embarked on my PhD journey. Since then, we have shared nearly every moment of this academic adventure — exploring Pittsburgh, traveling to different cities, and going to the office nearly every day. Through you, I have forged countless friendships and maintained both physical and mental well-being. While you may not understand a word of this thesis, I dedicate it to you.

This academic journey has been shaped and enriched by each individual mentioned here. Thank you all once again for your contributions to this significant chapter of my life :)

Table of contents

List of figures xii

List of tables xv

1 Introduction 1
- 1.1 Machine Learning & Programming . . . 3
  - 1.1.1 Unit Data Structure for Inductive Bias . . . 4
  - 1.1.2 Gradient Mapping of Unit . . . 5
  - 1.1.3 Interpretability: Data Influence Analysis . . . 5
  - 1.1.4 Controllability: Automated ML . . . 7
  - 1.1.5 Remarks . . . 8
- 1.2 Thesis Overview . . . 8
  - 1.2.1 Controllability: Automated ML . . . 9
  - 1.2.2 Interpretability: Data Influence Analysis . . . 9

2 Automatic Differentiation for Generalized Meta Learning 12
- 2.1 Introduction . . . 12
- 2.2 Background . . . 14
- 2.3 Method . . . 16
  - 2.3.1 Dataflow Graph for Generalized Meta Learning . . . 16
  - 2.3.2 Gradient Calculation with Best-Response Jacobians . . . 17
  - 2.3.3 Execution of Generalized Meta Learning . . . 18
- 2.4 Software Design . . . 18
- 2.5 Experiments . . . 20
  - 2.5.1 Differentiable Neural Architecture Search . . . 20
  - 2.5.2 Data Reweighting for Class Imbalance . . . 21
  - 2.5.3 Correcting & Reweighting Corrupted Labels . . . 23
  - 2.5.4 Domain Adaptation for Pretraining & Finetuning . . . 24

2.5.5 Design Choice Analysis . . . 25
2.6 Related Work . . . 27
2.7 Conclusion . . . 28

3 Scaling Meta Learning **29**
3.1 Introduction . . . 29
3.2 Background . . . 31
3.3 Scaling Meta Learning . . . 32
3.3.1 Base Jacobian Inverse . . . 33
3.3.2 Algorithmic Adaptation for Adaptive Optimizers . . . 35
3.3.3 Efficient Distributed Training & Implementation . . . 36
3.4 Experiments . . . 37
3.4.1 Noisy Finetuning of Large Language Models . . . 38
3.4.2 Continued Pretraining of Large Language Models . . . 40
3.4.3 Scale-Agnostic Efficient Data Pruning . . . 41
3.4.4 The Effect of Scaling in Model-Agnostic Meta Learning . . . 42
3.5 Related Work . . . 43
3.6 Conclusion . . . 44

4 ML Debugging Systems **46**
4.1 Introduction . . . 46
4.2 Scalability Bottlenecks in Influence Functions . . . 48
4.3 Scaling Data Valuation & Influence Functions . . . 50
4.3.1 Algorithm: Memory and Compute Efficient Gradient Projection . . . 50
4.3.2 Theory: Why Gradient Projection Works in Influence Functions . . . 51
4.3.3 Software: Compatibility, Extensibility, and Usability . . . 53
4.4 Experiments . . . 56
4.4.1 Quantitative Accuracy with Counterfactual Evaluation . . . 56
4.4.2 Scaling to Billion-Scale Models & Datasets . . . 57
4.5 Related Work . . . 76
4.6 Conclusion . . . 76

5 Conclusion **78**
5.1 Data Curation for Foundation Models . . . 78
5.2 Data Valuation & Attribution . . . 79

Appendix A Appendix for Chapter 3 81
A.1 Code Example . . . 81
A.1.1 Data Reweighting for Class Imbalance . . . 81
A.1.2 Correcting & Reweighting Corrupted Labels . . . 83
A.1.3 Domain Adaptation for Pretraining & Finetuning . . . 85
A.1.4 Differentiable Neural Architecture Search . . . 87
A.2 Experiment Details . . . 89
A.2.1 Data Reweighting for Class Imbalance . . . 89
A.2.2 Correcting & Reweighting Corrupted Labels . . . 89
A.2.3 Domain Adaptation for Pretraining & Finetuning . . . 90
A.2.4 Differentiable Neural Architecture Search . . . 91
A.3 Systems Support . . . 92
A.3.1 Differentiable Neural Architecture Search . . . 92
A.3.2 Data Reweighting for Class Imbalance . . . 92

Appendix B Appendix for Chapter 4 93
B.1 Philosophies behind SAMA & Scalable Meta Learning . . . 93
B.2 Experiment Details . . . 95
B.2.1 Noisy Finetuning of Large Language Models . . . 95
B.2.2 Continued Pretraining of Large Language Models . . . 95
B.2.3 Scale-Agnostic Efficient Data Pruning . . . 96

Appendix C Appendix for Chapter 5 98
C.1 Code Examples . . . 98
C.1.1 Log Extraction . . . 98
C.1.2 Influence Computation . . . 99
C.2 Experiment Details . . . 101
C.2.1 Quantitative Counterfactual Experiments . . . 101

References 103

List of figures

1.1 A diagram of comparing traditional programming and machine learning. . . 6

2.1 In `Engine` (left), users define their GML program as a hierarchy/graph of optimization problems. In `Problem` (middle), users define an optimization problem with a data loader, cost function, module, and optimizer, while upper/lower level constraint problems (*i.e.* $\mathscr{U}_k, \mathscr{L}_k$) are injected by `Engine`. The "step" function in `Problem` serves as the base of gradient-based optimization, abstracting the one-step gradient descent update process. Finally, users can easily try out different best-response Jacobian algorithms & system features (right) via `Config` in a modular manner. 13

2.2 Example GML dataflow graph. 16

2.3 Dataflow graphs for all our experiments 20

2.4 Convergence analysis of different best-response Jacobian algorithms on the synthetic hyperparameter optimization task 26

2.5 Convergence analysis of different best-response Jacobian algorithms on the data reweighting task 27

3.1 **Top:** Table showing a scalability comparison. **Bottom left:** Plot of throughput vs memory of different GBML algorithms on the *noisy finetuning of BERT-base* experiment. SAMA achieves better memory/compute efficiency overall given a fixed model, and the gap further widens by distributing compute across multiple GPUs with our efficient distributed training strategy. **Bottom right:** Plot of memory vs model size (*i.e.*, # of parameters) of different GBML algorithms on the *continued pretraining of RoBERTa* experiment. SAMA demonstrates the least significant increase in GPU memory usage with the increasing model size compared to baseline methods. 30

3.2 **Left:** $\cos(g_\lambda, g_{approx})$, where g_λ is a ground truth (i.e. closed-form) meta gradient, and g_{approx} is an approximate meta gradient obtained with various popular meta learning algorithms including SAMA. **Right:** $\|\lambda_t - \lambda^*\|_2$, where λ_t is the meta parameter λ after t meta updates. 34
3.3 The overall workflow of meta gradient computation with SAMA in the distributed data parallel setting. In detail, SAMA consists of three *first-order* backward passes performed with the underlying automatic differentiation engine, and one manual backward pass for algorithmic adaptation for the adaptive optimizer. Gradient synchronization is performed only once in the last backward pass with communication-computation overlap to minimize the communication bottleneck. 36
3.4 **Top Left:** ImageNet-1k data pruning results with ResNet-50. Reported numbers are relative accuracy compared to full training accuracy (*i.e.*, pruned_acc/full_acc). Accuracy for other baseline methods is obtained from DynaMS [162]. **Top Right:** CIFAR-10 data pruning results with ResNet-18. Accuracy for other baseline methods is obtained from Deepcore [59]. **Bottom:** Relative time spent in finding data to prune compared to full ImageNet-1k training time. . 42
3.5 Few-shot image classification accuracy on Omniglot 20-way 1-/5-shot tasks with varying network sizes. 43

4.1 Data valuation system architecture. **(Left Bottom)** We first extract the Hessian and gradients for all training data using efficient gradient projection LOGRA and store them in a database. **(Left Top)** At test time, we similarly extract gradients and query the database. **(Right)** The database returns similarity scores with respect to training examples that can be used for data valuation/attribution. 47
4.2 LOGRA. 51
4.3 Quantitative accuracy evaluation of data valuation algorithms. We excluded TRAK in the WikiText experiments due to lack of a public implementation for language modeling tasks. 56
4.4 Qualitative accuracy of data valuations with LOGRA. Important keywords in each example are *manually* highlighted for the improved readability. 59
4.5 Example 1. Llama3-8B-Instruct data valuation result. 60
4.6 Example 2. Llama3-8B-Instruct data valuation result. 61
4.7 Example 3. Llama3-8B-Instruct data valuation result. 62
4.8 Example 4. Llama3-8B-Instruct data valuation result. 63

4.9 Example 5. Llama3-8B-Instruct data valuation result. LoGra identifies novel literature as most valuable data. 64
4.10 Example 6. Llama3-8B-Instruct data valuation result (failure). 65
4.11 Example 1. GPT2-XL data valuation result. 66
4.12 Example 2. GPT2-XL data valuation result. 67
4.13 Example 3. GPT2-XL data valuation result. 68
4.14 Example 4. GPT2-XL data valuation result. 69
4.15 Example 5. GPT2-XL data valuation result (failure). 70
4.16 Example 1. Pythia-1.4B data valuation result. LoGra captures the broad topic of soccer but lacks the specificity (except for the third most valuable data, which states that Christiano Ronaldo is the best soccer play who won the Ballon d'Or award). 71
4.17 Example 2. Pythia-1.4B data valuation result. We suspect that the random url in the model output dominates the query gradient and affects the data valuation result. 72
4.18 Example 3. Pythia-1.4B data valuation result. 73
4.19 Example 4. Pythia-1.4B data valuation result. 74

List of tables

2.1 Supported features in BETTY 19
2.2 DARTS re-implementation results. AID-FD refers to implicit differentiation with a finite difference method, and * indicates the difference in the implementation of AID-FD explained above. 21
2.3 MWN experiment results. IF denotes an imbalance factor. AID-CG/NMN/FD respectively stand for implicit differentiation with conjugate gradient/Neumann series/finite difference. 22
2.4 MWN+BERT experiment results. fp32 and fp16 respectively stand for full-precision and mixed-precision training. 23
2.5 Wrench + MLP results. 24
2.6 Wrench + BERT-base results. 24
2.7 Clothing-1M + ResNet-50 results. 24
2.8 Domain Adaptation for Pretraining & Finetuning results. Reported numbers are classification accuracy on the target domain (right of arrow), after pretraining on the source domain (left of arrow). We note that *Baseline* is a two-layer, and *Baseline + Reweight* a three-layer, GML program. 25

3.1 WRENCH results. R and C in the first column stand for data reweighting and label correction operations. The number in parentheses indicates standard deviation for each experiment over 3 runs. 38
3.2 Memory and throughput analysis on AGNews with 4 V100 GPUs. 39
3.3 Ablation results on AGNews 39
3.4 Ablation results on IMDB 40
3.5 Experiment results for auxiliary learning with the continued pretraining task. Following [62], we report test micro-F1 for ChemProt and macro-F1 for the other datasets. The number in parentheses indicates the standard deviation for each experiment over 3 runs. 41

4.1 Memory & compute efficiency analyses for LOGRA and EKFAC. Throughput is measured as tokens/s for logging and (train, test) pairs/s for influence computations. * EKFAC logging consists of two subphases of KFAC fitting (left of /) and corrected eigenvalue fitting (right of /). 58

A.1 GPU memory usage analysis for DARTS. 92
A.2 GPU memory usage analysis for MWN with ResNet-50. 92

B.1 Hyperparameters for *noisy finetuning of large language models* experiments. 95
B.2 Hyperparameters for *continued pretraining of large language models* experiments. 96
B.3 Hyperparameters for *ImageNet-1k data pruning* experiments 96
B.4 Hyperparameters for *CIFAR-10 data pruning* experiments 96

C.1 Hyperparameter used in experiments in Section 4.4 101

Chapter 1

Introduction

The exponential progress in machine learning (ML) and artificial intelligence (AI) research has led to the widespread adoption of advanced ML systems, such as large language models (LLMs) [19, 157], across various sectors including the software industry [3, 139], digital art [71, 138], and science [7]. Nevertheless, as these systems increasingly integrate into our daily lives, a lack of principled understanding of how large-scale ML systems work poses significant challenges to both users and developers. This rapid deployment of new, not fully understood technologies can lead to various societal issues, from which ML systems are not immune [14]. Several examples highlight these challenges:

Copyright Infringement Recently, the New York Times (NYT) filed a lawsuit against OpenAI, alleging that ChatGPT was trained on NYT's copyrighted data [57]. While model providers like OpenAI claim that training ML models on copyrighted data is "fair use," it is well known that these models often regurgitate memorized training data [20], potentially providing a way to bypass the paywall system for accessing copyrighted data. This raises ethical questions about the extent to which ML models, generating revenue through the use of copyrighted data, should compensate original content creators. Identifying the specific training data utilized in proprietary LLMs to generate certain outputs remains a complex, unresolved issue [23].

Privacy and Data Protection Studies have shown that LLMs trained on vast internet-scale datasets can memorize and expose users' private information via hand-crafted prompts [123]. This aspect is particularly concerning in the context of stringent global data protection laws, such as GDPR [161] and CCPA [11]. Understanding the risk or mechanism of such memorization of private information in ML models and preventing it is thus of paramount importance in safeguarding user privacy [20].

Challenges in ML development Unintended behaviors in ML systems, such as incorrect predictions or biases, can stem from various causes [149, 156]. These might include the absence of similar samples in the training data or mislabeling within the dataset. Without effective mechanisms to swiftly pinpoint the root causes of such issues, rapid iteration and improvement in ML development become challenging.

To address these challenges, this thesis focuses on understanding ML systems through the lens of their *training data*. This approach is based on the premise that ML systems are largely reflections of the data they are trained on. For instance, if training data exhibit biases towards a specific subgroup of people [156] or includes private information [116], there is an increased chance that the ML model trained on such training data inherits similar biases or reveals private information in the deployment phase.

Consequently, this thesis begins by establishing a principled framework for understanding the influence of (each) training data on the final ML system using the counterfactual analysis: *how does including (or excluding) a specific training sample affect the optimization procedure and thereby the final model weights?* To answer this question, we can model the change in the training gradient when in/excluding a specific example at each time step, compute the change in the weight update due to the gradient change, and accumulate such changes over the entire optimization steps. More formally, we can view the gradient-based optimization process as dynamical systems or discretized differential equations [94, 98, 151]. Then, we can interpret training data as initial values, and study its impact on the model throughout the optimization process, drawing inspiration from well-established techniques in initial value problems [6]. When combined with other mathematical tools like Taylor series, these analyses enable applications such as data attribution and data optimization, both of which are important in interpreting and controlling ML systems.

While this approach offers theoretical appeal, it faces significant scalability challenges in practice, as the dimension of the optimization problem is equal to the number of model parameters, which often exceeds billions in recent ML systems [19, 157]. To address this, this thesis combines knowledge from both algorithms and systems to mitigate the computational and memory costs of the analysis. Specifically, we introduce efficient approximations to existing analysis methods that are compatible with modern hardware (*e.g.*, GPUs) and distributed computing. These innovations enable the application of our framework to large-scale ML models, bridging the gap between theoretical understanding and practical implementation.

Finally, to ensure that our mathematical frameworks for understanding training data has tangible real-world impacts, we further endeavor to operationalize our frameworks from traditional software perspectives. More specifically, we draw an insightful analogy between ML and traditional programming inspired by the long-established concept of inductive programming [44], especially noting that training data is a major component in inductive biases. Such an approach proves particularly enlightening, as questions challenging to address in the context of ML (*e.g.*, interpretability) oftentimes become clearer when posed in the realm of traditional programming. Additionally, the rich history of established practices in traditional programming, like unit testing and version control, offers a valuable repository of strategies potentially adaptable to ML development.

While our approach shares similarities with other interpretability research, it offers a unique perspective. Notably, mechanistic interpretability [39, 115] parallels model weights in ML to binary machine code in conventional software, aiming to develop reverse-engineering techniques for ML inspired by those used in traditional programming. Despite promising initial results, machine code often proves too low-level for most practitioners to work with comfortably, thereby failing to provide actionable insights to improve their models. In

contrast, this thesis primarily focuses on the "source code" analogue in ML, aiming to develop a framework for understanding ML models at a more accessible and conceptual level. This approach bridges the gap between theoretical understanding and practical application, offering insights that can directly inform model development and improvement.

1.1 Machine Learning & Programming

Towards this goal, we review basic concepts in traditional programming (or software development) and machine learning respectively. The primary goal of software development is to create executable software that fulfills the needs of its stakeholders. This process begins with *specifications*, which outline the desired functionalities, performance criteria, and operational requirements of the final software in a highly abstract form. Developers then translate these abstract specifications into concrete rules and instructions by writing *source code* with the help of programming language that provides the necessary syntax and semantics. Subsequently, source code is translated into binary *machine code* with *compilers*, rendering the resulting software directly executable by the computer. In essence, the software development process can be viewed as a gradual transformation of high-level, human-friendly abstractions into low-level, machine-friendly instructions.

Machine learning (ML) conducts model fitting over diverse tasks, such as object detection, image classification, or language modeling, given *inductive biases* from developers. This process can be formulated as an *optimization* problem, as shown in the following equation:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ L(\theta; D, A, \lambda) \tag{1.1}$$

where θ / θ^* are the trainable / final model weights, and (D, A, L, λ) respectively denote the training dataset, network architecture, objective, and hyperparameters, together which define inductive biases in ML. Oftentimes, under the independent and identically distributed (*i.i.d.*) data and negative log-likelihood assumptions, Eq. (1.1) is further decomposed into:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \frac{1}{n}\sum_{i=1}^{n} L_i(\theta; x_i, A_i, \lambda_i) \tag{1.2}$$

where $(x_i, A_i, L_i, \lambda_i)$ are i-th training data, architecture, objective, and hyperparameter. We note that, while (A_i, L_i, λ_i) are commonly fixed across data samples, there are several cases like multi-task learning where they may vary.

Among multiple approaches to solving the optimization problem in Eq. (1.2), the most popular approaches, especially in the context of deep neural networks, are stochastic gradient descent (SGD) and its variants. These approaches iteratively update the model weights using gradients computed from each mini-batch until the *final model weights* θ^* are obtained at

convergence, as illustrated in the update equation below:

$$\theta_{t+1} = \theta_t - F_t\left(\frac{\partial \frac{1}{|B_t|}\sum_{i_k \in B_t} L_{i_k}(\theta; A_{i_k}, x_{i_k})}{\partial \theta}\bigg|_{\theta=\theta_t}\right), \quad \theta^* = \lim_{t\to\infty}\theta_t \tag{1.3}$$

where B_t is a mini-batch and F_t is the update rule of optimizer at time step t.

Based on the concepts introduced above, inductive programming likens inductive biases (D, A, L, λ) in ML to specifications/source code in traditional software,[1] 2) the optimization process in Eq. (1.3) to the compilation process, and 3) the final model weight θ^* to machine code.[2] Despite the appeal of this analogy, we emphasize the obscure interpretation of ML inductive biases as both source code and specifications. In traditional software, source code implements explicit instructions and rules for the computer to follow, serving as a basis for deductive reasoning to understand software behavior. In ML, inductive biases implement the desired behaviors as source code, but in a highly abstract manner as specifications. In this thesis, we argue that the specification nature of inductive biases substantially complicates the process of understanding their influence on ML model behaviors, in contrast to the more direct impact of source code on traditional software behaviors.

1.1.1 Unit Data Structure for Inductive Bias

One common strategy to improve the clarity, maintenance, and organization of source code in traditional programming is introducing a custom data structure (or composite data type), such as "struct" in C or "dataclass" in Python. Inspired by this strategy, we propose a unique "unit" data structure for ML inductive biases that would serve as a foundation for understanding their influence on final model behaviors. To this end, we first notice that the optimization problem in Eq. (1.2) is a summation of n sub-objectives, potentially providing the necessary modularity for controlling and interpreting ML models. Accordingly, we define a unit data structure for ML as follows:

Definition 1 *A unit of the inductive bias in machine learning is defined as* $(x_i, L_i, A_i, \lambda_i)$.

This conceptualization, where x_i, L_i, A_i, and λ_i are considered collectively, is important as changing one component of this tuple can significantly affect the following optimization (compilation) process and model weights (machine code). For example, keeping data x_i and architecture A_i constant, but changing L_i to its adversarial counterpart $-L_i$, could drastically modify the final behavior of the ML model. Similarly, even if x_i and L_i are fixed, if we update different parts of the model (*e.g.*, last layer finetuning vs full finetuning), they may result in very different ML models. As a result, this sensitivity to context challenges the stability of these elements as basic units for analyzing model behaviors. Importantly, we note that the concept of our unit data structure is not limited to source code, but can also be applied in the

[1] Inductive biases in ML are originally considered as incomplete specifications in inductive programming, but a recent paradigm like Software 2.0 interprets it as source code

[2] Interestingly, under the "training data as source code" analogy, claims of models like LLaMA and Mistral being open source are challenged, as their training data are not publicly accessible.

test phase, given test data, architecture, and objective. In essence, ML development — from data preparation to architecture and objective design — amounts to writing source code by instantiating multiple abstract unit variables.

1.1.2 Gradient Mapping of Unit

While the modularity offered by Eq. (1.2) and our proposed unit data structure enhances the organization of ML inductive biases, it does not necessarily provide explicit rules or instructions for the final model to follow, unlike traditional source code. To bridge this gap, we instead propose to map these unit variables into a vector space, following the concept of *distributed representation*, and build the algebraic (or mathematical) structure that can serve as a ground for deductive reasoning to understand their influence on final model behaviors. Here, we specifically consider mapping each unit variable to its gradient computed at the final model weight θ^*.

Definition 2 *Gradient mapping* $g : S \to V \subset \mathbb{R}^{\dim(\theta)}$ *of the unit variable* (x', L', A', λ') *is:*

$$g\big((x', L', A')\big) = \frac{\partial L'(\theta; x', A', \lambda')}{\partial \theta}\Big|_{\theta=\theta^*}$$

This mapping process enables us to translate the abstract nature of an inductive bias unit into a more concrete and analyzable form.

Next, we incorporate the Mahalanobis distance with respect to the training gradient distribution as our chosen metric in the vector space V. To set the stage for this, two key observations are noted. First, the average training gradient μ at the final model weight θ^* is 0, as a result of the convergence condition in optimization. Second, under the maximum likelihood assumption, the covariance of training gradient, which is the Fisher Information Matrix (FIM) by definition[3], equates to the Hessian. Based on these observations, we define an inner product that leads to the above Mahalanobis metric as follows:

Definition 3 *For* $g', g'' \in V$*, inner product can be defined as:*

$$inner(g', g''; D, L, A) = g''^T F^{-1} g' = g''^T H^{-1} g'$$

where F and H are respectively the FIM (*i.e.*, $\mathbb{E}_{g \sim D,L,A}\left[gg^T\right]$) and the Hessian. Extending the above definition of an inner product, one can easily derive other measures including norm, distance, or cosine similarity, and thereby enables diverse mathematical analyses.

1.1.3 Interpretability: Data Influence Analysis

Interestingly, we notice that the inner product defined in Definition 3 corresponds to the *influence function* commonly referenced in the robust statistics literature [27, 29, 96, 28,

[3]Empirical FIM, which is computed with gradient sampled from empirical distribution, isn't same with Hessian. Nevertheless, empirical FIM is frequently used instead of true FIM in practice without noticeable differences in performance.

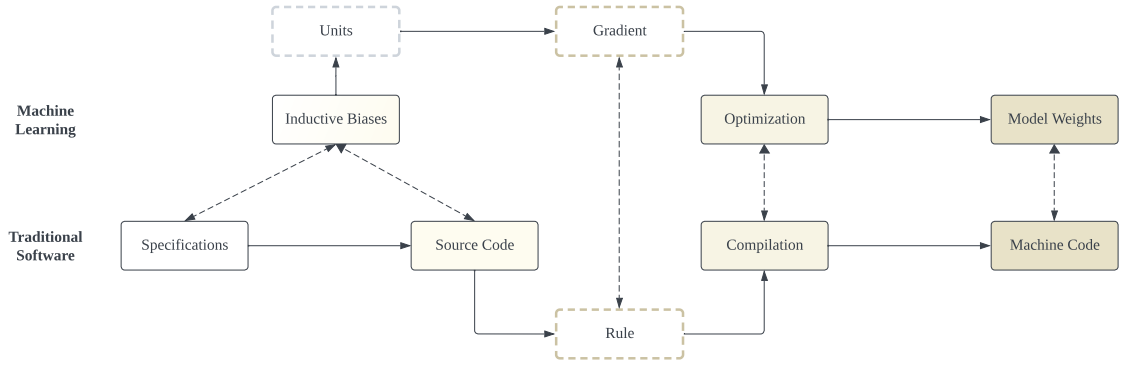


Fig. 1.1 A diagram of comparing traditional programming and machine learning.

87]. In simpler terms, the influence function is an approximate measure of how much an estimator depends on a specific observation, based on a Taylor series expansion, or, more formally, Cauchy's implicit function theorem. In the context of our research, this translates to approximately measuring the dependence of a test unit variable on a specific train unit variable. For instance, if the inner product (or influence score) between train and test unit variables is high (or low), meaning their gradients are positively (or negatively) aligned, then the train unit variable increases (or decreases) the likelihood of the test unit variable. Conceptually, this is akin to the concrete rules and actions implemented in source code of traditional software. Furthermore, if we linearly combine inner product values (or influence scores) of all train unit variables as in Eq. (1.2), ML development can be understood as writing multiple rules by instantiating various train unit variables.

Furthermore, we notice that the norm of each unit variable, *i.e.* $inner(g',g') = g'^T H^{-1} g'$, equates with the *score* in statistical hypothesis testing, a measure that has been employed for out-of-distribution (OOD) detection [15]. Alternatively, one can also interpret the norm of each unit variable as an uncertainty measure under the Laplace approximation of the Hessian.

Theorem 1 *Given p_{LA} is the Laplace approximation of posterior, the following holds:*

$$inner(g',g') = \mathbb{E}_{\theta\sim p_{LA}}\left[\left(\frac{\partial L'(\theta;A',x')}{\partial\theta}\right)^T \cdot \left(\frac{\partial L'(\theta;A',x')}{\partial\theta}\right)\right] \approx Var_{\theta\sim p_{LA}}\left[L(\theta;x',L',A')\right]$$

The immediate application is that we can build the "try-except" (or "try-catch") statement by utilizing the norm as an indicator for the improved runtime error handling.

Moreover, interpreting the norm as a measure of uncertainty for each unit variable allows us to refine our understanding of the influence concept from Definition 3. Intuitively, when there is greater uncertainty associated with a specific train unit variable, the inner product computed against it would better be penalized accordingly. Two straightforward penalty strategies include: subtracting the uncertainty from the influence, or dividing the influence by the uncertainty. Mathematically, each of these strategies corresponds to computing the L_2 distance and the cosine similarity. Notably, the latter method, dividing the influence

by the uncertainty, aligns with the concept of "RelatIF" that has been explored in prior literature [12].

As our framework establishes a connection between ML and traditional software development at a functional level, unlike the more conceptual approach seen in inductive programming paradigms (*e.g.*, Software 2.0), it naturally leads to the adoption of various software development practices in the ML context. For example, developers can create unit tests by selecting a small number of test unit variables that embody desired characteristics (*e.g.*, fairness). They can then evaluate the effectiveness of their ML source code implementation by computing the inner product of all train unit variables against these test unit variables. This would further pave the way for integrating concepts including version control, continuous integration and continuous deployment (CI/CD), and monitoring into ML development. These practices are vital for controlling, interpreting, and efficiently managing complex ML systems, mirroring their critical role in traditional software development. By borrowing and adapting these established software practices, we can enhance controllability and interpretability of ML models, and thereby enable agile ML development, especially under the evolving requirements.

1.1.4 Controllability: Automated ML

In the previous section, we showed that the correctness of the source code implementation can be measured by computing inner product against test unit variables. A natural next step is to automatically modify source code in a way that modified source code maximizes the inner product against test unit variables. This process is highly related to program synthesis, a concept of automatically constructing source code, of which resulting software (provably) meets high-level specifications. In this section, we derive the formal connection between these two concepts.

Reminding that inductive biases in ML serve as both specifications and source code, we can understand ML program synthesis as solving train inductive biases, of which resulting model weights θ^* after optimization, satisfies test inductive biases. To ensure modifiablility of train inductive biases, we posit they are additionally parameterized by hyperparameters λ. Given this setup, ML program synthesis can naturally be formulated as bi-level optimization or meta learning as in the following equations:[4]

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, L_{test}(\theta^*(\lambda); D_{test}) \tag{1.4}$$

$$s.t.\ \ \theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}}\, L_{train}(\theta; D_{train}, A_{train}, \lambda) \tag{1.5}$$

One of the popular approaches to solve meta learning is *implicit differentiation*, which calculates the best-response Jacobian by leveraging Cauchy's Implicit Function Theorem (IFT) and re-interpreting the base optimization problem from the perspective of fixed-point

[4]Meta learning, as its name indicates, shares a lot of common concepts with meta-programming, of which sub-field is program synthesis.

iteration as follows:

$$\frac{\partial L_{test}}{\partial \lambda} = \underbrace{\frac{\partial \theta^*}{\partial \lambda}}_{\text{best-response Jacobian}} \cdot \underbrace{\frac{\partial L_{test}}{\partial \theta^*}}_{\text{direct gradient}} = -\frac{\partial^2 L_{train}}{\partial \lambda \partial \theta^*} \cdot \frac{\partial^2 L_{train}}{\partial \theta^{*2}} \cdot \frac{\partial L_{test}}{\partial \theta^*} \tag{1.6}$$

Assuming that the Hessian is independent from hyperparameters λ, Eq. (1.6) can be re-written as:

$$\frac{\partial L_{test}}{\partial \lambda} = -\frac{\partial}{\partial \lambda}\left(\frac{\partial L_{train}}{\partial \theta^*} \cdot \frac{\partial^2 L_{train}}{\partial \theta^{*2}} \cdot \frac{\partial L_{test}}{\partial \theta^*}\right) \tag{1.7}$$

Since the term inside the parentheses is inner product defined in Definition 2.4, we showed that ML program synthesis by solving Eq. (1.5) with the update rule of Eq. (1.7) to convergence is equal with maximally aligning gradients between train and test unit variables, thereby gaining controllability.

1.1.5 Remarks

Before concluding this section, it is important to acknowledge that the concepts of gradient mapping and Mahalanobis distance in our framework are not entirely novel. For instance, the work of Jaakkola et al. on the Fisher kernel, which aimed to utilize generative models in discriminative classifiers, shares similarities with our approach [79]. However, such frameworks have not been extensively explored in modern ML contexts, primarily due to scalability challenges arising from the high-dimensional nature of gradients in contemporary models. Our thesis contributes to this area by addressing these scalability concerns and further extends the understanding of modern ML models. Additionally, an attempt to improve controllability and interpretability of ML models by drawing the parallels between ML and software development is an original contribution of this thesis. This not only leverages the functional aspects of software engineering but also adapts them to the unique needs and challenges of ML, offering new perspectives and methodologies in the field.

1.2 Thesis Overview

In main chapters, we describe our efforts in building practical applications of aforementioned automated ML and data influence analyses, including but not limited to data optimization, training data attribution, and hallucination detection. From the technical viewpoint, main challenges involve scalability, algorithmic instability, and programming interface, due to the inherently high-dimensional and stochastic nature of a gradient space. We attempt to tackle these issues holistically by approaching the problems from both ML algorithm and system perspectives. More detailed contributions are provided below.

1.2.1 Controllability: Automated ML

Systems (Section 2 & [24]): Given the specific objective, optimizing ML source code amounts to aligning gradient in the intermediate representation space, which shares the same mathematical formulation as gradient-based bi-level optimization or meta learning (GBML). However, many GBML algorithms are notoriously difficult to implement and memory/compute intensive, mostly due to Hessian-involved operations. As an initial step towards closing this gap, we introduce BETTY, an automatic differentiation library for generalized meta learning (or multi-level optimization) that hides the intricate mechanism of gradient alignment (or meta gradient computation) from the users while supporting various system optimization for the improved scalability. We empirically demonstrate that BETTY can be used to optimize an array of ML source code, while also observing up to 11% increase in test accuracy, 14% decrease in GPU memory usage, and 20% decrease in training wall time over existing implementations on multiple benchmarks.

Algorithms (Section 3 & [22]): To mitigate the scalability and stability issues stemming from the Hessian-involved operations in GBML, we develop SAMA, a variant of an implicit differentiation algorithm. Specifically, SAMA is designed to flexibly support a broad range of adaptive optimizers in the base level of meta learning programs, while reducing computational burden by avoiding explicit computation of second-order gradient information, and exploiting efficient distributed training techniques implemented for first-order gradients. Evaluated on multiple large-scale meta learning benchmarks, SAMA showcases up to 1.7/4.8$\times$ increase in throughput and 2.0/3.8$\times$ decrease in memory consumption respectively on single-/multi-GPU setups compared to other baseline meta learning algorithms. BETTY is publicly available at https://github.com/leopard-ai/betty.

1.2.2 Interpretability: Data Influence Analysis

Systems (Section 4 & [23]): Interpreting the influence of each training data on the final model behavior can be achieved by comparing train and test gradients using influence functions. To operationalize this process, we use gradient as a representation for each train/test example and translate the data influence analysis as the representation similarity analysis, for which various system optimizations (*e.g.*, vector database [83]) exist. Specifically, we introduce a software package called LOGIX that intercepts per-sample gradients, computes necessary statistics (*e.g.*, Fisher information matrix), and write/query gradients to vector database. Our software design, which is similar to those of ML logging tools like Tensorboard and Weights & Biases, achieves high compatibility & interoperability with diverse (scalability) tools and frameworks in the recent ML ecosystem, such as Deepspeed [133] and Transformers [166], and thereby enables easy *conversion* of users' existing training codes into data influence analysis codes. Finally, LOGIX implements a few additional system optimizations such as asynchronous data IO to improve efficiency of data influence analyses. LOGIX is publicly available at https://github.com/logix-project/logix.

Algorithms (Section 4 & [23]): Due to the high-dimensional nature of gradient, the data influence analysis significantly suffers from high compute, memory, and storage costs. To address this issue, we develop an efficient gradient projection strategy called LOGRA that leverages the gradient structure in backpropagation. We then provide a theoretical motivation of gradient projection approaches to influence functions to promote trust in the data influence analysis. In our data valuation experiments, LOGRA, implemented with LOGIX, achieves competitive accuracy against more expensive baselines while showing up to 6,500× improvement in throughput and 5× reduction in GPU memory usage when applied to Llama3-8B- Instruct and the 1B-token dataset.

Controllability: Automated ML

Chapter 2

Automatic Differentiation for Generalized Meta Learning

Automated ML, often formulated as meta learning or bi-level optimization, enables users to control the final behavior of ML models by letting them curate a small number of test unit variables that possess desired characteristics (*e.g.*, fairness and label correctness). However, meta gradient (or hypergradient) in Eq. (1.7) is notoriously difficult to implement and memory/compute intensive. In this section, we introduce BETTY, an automatic differentiation framework for generalized meta learning (GML), that abstracts away implementation details of meta gradient algorithms as well as provides the foundation for various systems support for scalability (*e.g.*, mixed precision, distributed training) for developers.

2.1 Introduction

Generalized meta learning (GML) addresses nested optimization scenarios, where upper level optimization problems are constrained by lower level optimization problems following an underlying hierarchical dependency. GML has gained considerable attention as a unified mathematical framework for studying diverse problems including meta-learning [43, 130], hyperparameter optimization [45], neural architecture search [99], and reinforcement learning [89, 131]. While a majority of existing work is built upon bilevel optimization, the simplest case of GML, there have been recent efforts that go beyond this two-level hierarchy. For example, [128] proposed trilevel optimization that combines hyperparameter optimization with two-level pretraining and finetuning. More generally, conducting joint optimization over machine learning pipelines consisting of multiple models and hyperparameter sets can be approached as deeper instances of GML [50, 128, 148, 152].

Following its increasing popularity, a multitude of optimization algorithms have been proposed to solve GML. Among them, *gradient-based (or first-order)* approaches [121, 103, 128, 141] have recently received the limelight from the machine learning community, due to their ability to carry out efficient high-dimensional optimization, under which all of the above listed applications fall. Nevertheless, research in gradient-based GML has been largely impeded by two major bottlenecks. First, implementing gradients in generalized

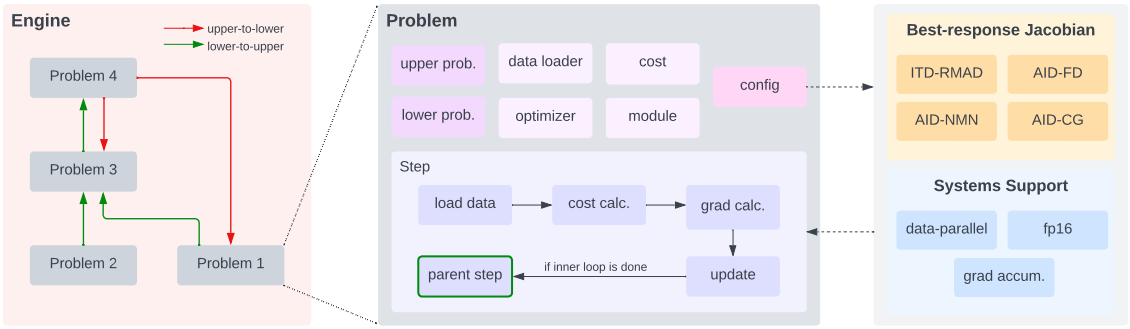


Fig. 2.1 In `Engine` (left), users define their GML program as a hierarchy/graph of optimization problems. In `Problem` (middle), users define an optimization problem with a data loader, cost function, module, and optimizer, while upper/lower level constraint problems (*i.e.* $\mathscr{U}_k, \mathscr{L}_k$) are injected by `Engine`. The "step" function in `Problem` serves as the base of gradient-based optimization, abstracting the one-step gradient descent update process. Finally, users can easily try out different best-response Jacobian algorithms & system features (right) via `Config` in a modular manner.

meta learning, which is achieved by composing best-response Jacobians via the chain rule, requires both programming and mathematical proficiency. Second, algorithms for best-response Jacobian calculation, such as iterative differentiation (ITD) or approximate implicit differentiation (AID) [53], are memory and compute intensive, as they require multiple forward/backward computations and oftentimes second-order gradient (*i.e.* Hessian) information.

In recent years, there has been some work originating in the meta-learning community on developing software libraries that target some aspects of gradient-based GML [17, 32, 54]. For example, *JAXopt* [17] provides efficient and modular implementations of AID algorithms by letting the user define a function capturing the optimality conditions of the problem to be differentiated. However, *JAXopt* fails to combine the chain rule with AID to support general GML programs beyond a two-level hierarchy. Similarly, *higher* [54] provides several basic primitives (*e.g.* making PyTorch's [118] native optimizers differentiable) for implementing ITD/AID algorithms, but users still need to manually implement complicated internal mechanisms of these algorithms as well as the chain rule to implement a given instance of GML. Furthermore, most existing libraries do not have systems support, such as mixed-precision and data-parallel training, that could mitigate memory and computation bottlenecks. As a result, gradient-based GML research built upon these libraries has been largely limited to simple bilevel optimization and small-scale setups.

In this work, we attempt to bridge this gap between research and software systems by introducing BETTY, an easy-to-use and modular automatic differentiation library with various systems support for large-scale GML. The main contributions of this work are as follows:

1. We develop an efficient automatic differentiation technique for GML based on a novel interpretation of GML as a special type of dataflow graph (Section 2.3). In detail, gradient calculation for each optimization problem is automatically carried out by iteratively

multiplying best-response Jacobians (defined in Section 2.2) through the chain rule while reverse-traversing specific paths of this dataflow graph. This reverse-traversing procedure is crucial for efficiency, as it reduces the computational complexity of our automatic differentiation technique from $\mathcal{O}(d^3)$ to $\mathcal{O}(d^2)$, where d is the dimension of the largest optimization problem in the GML program.

2. We introduce a software library for GML, BETTY, built upon the above automatic differentiation technique. Our software design (Section 2.4), motivated by the dataflow graph interpretation, provides two major benefits: (1) it allows for incorporating various systems support, such as mixed-precision and data-parallel training, for large-scale GML, and (2) it facilitates implementation of GML programs of arbitrary complexity while allowing a modular interface for diverse algorithmic and systems design choices. The overall software architecture of BETTY is presented in Figure 2.1.

3. We empirically demonstrate that BETTY can be used to implement an array of GML applications with varying scales and complexities (Section 2.5). Interestingly, we observe that trying out different best-response Jacobian algorithms with our modular interface (which only requires changing one line of code) can lead to up to 11% increase in test accuracy, 14% decrease in GPU memory usage, and 20% decrease in training wall time on various benchmarks, compared with the original papers' implementations. Finally, we showcase the scalability of BETTY to models with hundreds of millions of parameters by performing GML on the BERT-base model with the help of BETTY's systems support, which was otherwise infeasible.

2.2 Background

To introduce GML, we first define a core concept known as a "constrained problem" [160].

Definition 4 *An optimization problem P is said to be **constrained** by λ when its cost function $\mathscr{C}$ has λ as an argument in addition to the optimization parameter θ (i.e. P : $\arg\min_\theta \mathscr{C}(\theta, \lambda, \cdots)$).*

Generalized meta learning [112] refers to a field of study that aims to solve a nested set of optimization problems defined on a sequence of so-called *levels*, which satisfy two main criteria: **A1)** upper-level problems are constrained by the *optimal* parameters of lower-level problems while **A2)** lower-level problems are constrained by the *nonoptimal* parameters of

upper-level problems. Formally, an n-level GML program can be written as:

$$
\begin{aligned}
P_n: \quad & \theta_n^* = \underset{\theta_n}{\operatorname{argmin}}\, \mathscr{C}_n(\theta_n, \mathscr{U}_n, \mathscr{L}_n; \mathscr{D}_n) && \triangleright \text{ Level } n \text{ problem} \\
& \ddots \\
P_k: \quad & \text{s.t. } \theta_k^* = \underset{\theta_k}{\operatorname{argmin}}\, \mathscr{C}_k(\theta_k, \mathscr{U}_k, \mathscr{L}_k; \mathscr{D}_k) && \triangleright \text{ Level } k \in \{2, \dots, n-1\} \text{ problem} \\
& \ddots \\
P_1: \quad & \text{s.t. } \theta_1^* = \underset{\theta_1}{\operatorname{argmin}}\, \mathscr{C}_1(\theta_1, \mathscr{U}_1, \mathscr{L}_1; \mathscr{D}_1) && \triangleright \text{ Level 1 problem}
\end{aligned}
$$

where, P_k stands for the level k problem, θ_k / θ_k^* for corresponding nonoptimal / optimal parameters, and $\mathscr{U}_k$ / $\mathscr{L}_k$ for the sets of constraining parameters from upper / lower level problems. Here, $\mathscr{D}_k$ is the training dataset, and $\mathscr{C}_k$ indicates the cost function. Due to criteria **A1** & **A2**, constraining parameters from upper-level problems should be nonoptimal (*i.e.* $\mathscr{U}_k \subseteq \{\theta_{k+1}, \cdots, \theta_n\}$) while constraining parameters from lower-level problems should be optimal (*i.e.* $\mathscr{L}_k \subseteq \{\theta_1^*, \cdots, \theta_{k-1}^*\}$). Although we denote only one optimization problem per level in the above formulation, each level could in fact have multiple problems. Therefore, we henceforth discard the concept of level, and rather assume that problems $\{P_1, P_2, \cdots, P_n\}$ of a general GML program are topologically sorted in a "reverse" order (*i.e.* P_n / P_1 denote uppermost / lowermost problems).

For example, in hyperparameter optimization formulated as bilevel optimization, hyperparameters and network parameters (weights) correspond to upper and lower level parameters (θ_2 and θ_1). Train / validation losses correspond to $\mathscr{C}_1$ / $\mathscr{C}_2$, and validation loss is dependent on optimal network parameters θ_1^* obtained given θ_2. Thus, constraining sets for each level are $\mathscr{U}_1 = \{\theta_2\}$ and $\mathscr{L}_2 = \{\theta_1^*\}$.

In this work, we focus in particular on *gradient-based* GML, rather than zeroth-order methods like Bayesian optimization [31], in order to efficiently scale to high-dimensional problems. Essentially, gradient-based GML calculates gradients of the cost function $\mathscr{C}_k(\theta_k, \mathscr{U}_k, \mathscr{L}_k)$ with respect to the corresponding parameter θ_k, with which gradient descent is performed to solve for optimal parameters θ_k^* for every problem P_k. Since optimal parameters from lower level problems (*i.e.* $\theta_l^* \in \mathscr{L}_k$) can be functions of θ_k (criterion **A2**), $\frac{d\mathscr{C}_k}{d\theta_k}$ can be expanded using the chain rule as follows:

$$
\frac{d\mathscr{C}_k}{d\theta_k} = \underbrace{\frac{\partial \mathscr{C}_k}{\partial \theta_k}}_{\text{direct gradient}} + \sum_{\theta_l^* \in \mathscr{L}_k} \underbrace{\frac{d\theta_l^*}{d\theta_k}}_{\text{best-response Jacobian}} \times \underbrace{\frac{\partial \mathscr{C}_k}{\partial \theta_l^*}}_{\text{direct gradient}} \tag{2.1}
$$

While calculating direct gradients (purple) is straightforward with existing automatic differentiation engines like PyTorch [118], a major difficulty in gradient-based GML lies in best-response Jacobian[1] (orange) calculation, which will be discussed in depth in Section 2.3. Once gradient calculation for each level k is enabled via Equation (2.1), gradient-based

[1]We abuse the term Jacobian for a total derivative here while it is originally a matrix of partial derivatives

optimization is executed from lower to upper level problems in a topologically reverse order, reflecting underlying hierarchies.

2.3 Method

While Equation (2.1) serves as a mathematical basis for gradient-based generalized meta learning, how to automatically and efficiently carry out such gradient calculation has not been extensively studied and incorporated into a software system that can support GML programs involving many problems with complex dependencies. In this section, we discuss the challenges in building an automatic differentiation library for GML, and provide solutions to address these challenges.

2.3.1 Dataflow Graph for Generalized Meta Learning

One may observe that the best-response Jacobian term in Equation (2.1) is expressed with a *total derivative* instead of a partial derivative. This is because θ_k can affect θ_l^* not only through a direct interaction, but also through multiple indirect interactions via other lower-level optimal parameters. For example, consider the four-problem GML program illustrated in Figure 2.2. Here, the parameter of Problem 4 (θ_{p_4}) affects the optimal parameter of Problem 3 ($\theta_{p_3}^*$) in two different ways: 1) $\theta_{p_4} \to \theta_{p_3}^*$ and 2) $\theta_{p_4} \to \theta_{p_1}^* \to \theta_{p_3}^*$. In general, we can expand the best-response Jacobian $\frac{d\theta_l^*}{d\theta_k}$ in Equation (2.1) by applying the chain rule for all paths from θ_k to θ_l^* as

$$\frac{d\mathscr{C}_k}{d\theta_k} = \frac{\partial \mathscr{C}_k}{\partial \theta_k} + \sum_{\theta_l^* \in \mathscr{L}_k} \sum_{q \in \mathscr{Q}_{k,l}} \left(\underbrace{\frac{\partial \theta_{q(1)}^*}{\partial \theta_k}}_{\text{upper-to-lower}} \times \left(\prod_{i=1}^{\text{len}(q)-1} \underbrace{\frac{\partial \theta_{q(i+1)}^*}{\partial \theta_{q(i)}^*}}_{\text{lower-to-upper}} \right) \times \frac{\partial \mathscr{C}_k}{\partial \theta_l^*} \right) \tag{2.2}$$

where $\mathscr{Q}_{k,l}$ is a set of paths from θ_k to θ_l^*, and $q(i)$ refers to the index of the i-th problem in the path q with the last point being θ_l^*. Replacing a total derivative term in Equation (2.1) with a product of partial derivative terms using the chain rule allows us to ignore indirect interactions between problems, and only deal with direct interactions.

To formalize the path finding problem, we develop a novel dataflow graph for GML. Unlike traditional dataflow graphs with no predefined hierarchy among nodes, a dataflow graph for generalized meta learning has two different types of directed edges stemming from criteria **A1** & **A2**: *lower-to-upper* and *upper-to-lower*. Each of these directed edges is respectively depicted with green and red arrows in Figure 2.2. Essentially, a lower-to-upper edge represents the directed dependency between two optimal parameters (*i.e.* $\theta_{P_i}^* \to \theta_{P_j}^*$ with $P_i < P_j$), while an upper-to-lower edge represents the directed dependency between nonoptimal and optimal parameters (*i.e.* $\theta_{P_i} \to \theta_{P_j}^*$ with $P_i > P_j$). Since we need to find paths from the

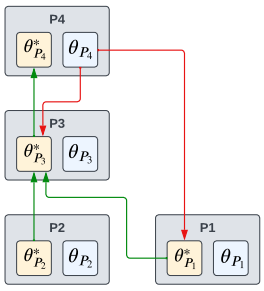


Fig. 2.2 Example GML dataflow graph.

nonoptimal parameter θ_k to the optimal parameter θ_l^*, the first directed edge must be an upper-to-lower edge (red), which connects θ_k to some lower-level optimal parameter. Once it reaches the optimal parameter, it can only move through optimal parameters via lower-to-upper edges (green) in the dataflow graph. Therefore, every valid path from θ_k to θ_l^* will start with an upper-to-lower edge, and then reach the destination only via lower-to-upper edges. The best-response Jacobian term for each edge in the dataflow graph is also marked with the corresponding color in Equation (2.2). We implement the above path finding mechanism with a modified depth-first search algorithm in BETTY.

2.3.2 Gradient Calculation with Best-Response Jacobians

Automatic differentiation for GML can be realized by calculating Equation (2.2) for each problem P_k ($k = 1, \cdots, n$). However, a naive calculation of Equation (2.2) could be computationally onerous as it involves multiple matrix multiplications with best-response Jacobians, of which computational complexity is $\mathcal{O}(d^3)$, where d is the dimension of the largest optimization problem in the GML program. To alleviate this issue, we observe that the rightmost term in Equation (2.2) is a vector, which allows us to reduce the computational complexity of Equation (2.2) to $\mathcal{O}(d^2)$ by iteratively performing matrix-vector multiplication from right to left (or, equivalently, reverse-traversing a path q in the dataflow graph). As such, matrix-vector multiplication between the best-response Jacobian and a vector serves as a base operation of efficient automatic differentiation for GML. Mathematically, this problem can be simply written as follows:

$$\text{Calculate} \quad \frac{\partial w^*(\lambda)}{\partial \lambda} \times v \tag{2.3}$$

$$\text{Given} \quad w^*(\lambda) = \underset{w}{\operatorname{argmin}}\, \mathcal{C}(w, \lambda). \tag{2.4}$$

Two major challenges in the above problems are: 1) approximating the solution of the optimization problem (*i.e.* $w^*(\lambda)$), and 2) differentiating through the (approximated) solution.

In practice, an approximation of $w^*(\lambda)$ is typically achieved by unrolling a small number of gradient steps, which can significantly reduce the computational cost [45]. While we could potentially obtain a better approximation of $w^*(\lambda)$ by running gradient steps until convergence, this procedure alone can take a few days (or even weeks) when the underlying optimization problem is large-scale [33, 35].

Once $w^*(\lambda)$ is approximated, matrix-vector multiplication between the best-response Jacobian $\frac{dw^*(\lambda)}{d\lambda}$ and a vector v is popularly obtained by either iterative differentiation (ITD) or approximate implicit differentiation (AID) [53]. This problem has been extensively studied in bilevel optimization literature [43, 45, 103], and we direct interested readers to the original papers, as studying these algorithms is not the focus of this work. In BETTY, we provide implementations of several popular ITD/AID algorithms which users can easily plug-and-play for their GML applications. Currently available algorithms within BETTY include ITD with reverse-mode automatic differentiation (ITD-RMAD) [43], AID with Neumann

series (AID-NMN) [103], AID with conjugate gradient (AID-CG) [130], and AID with finite difference (AID-FD) [99].

2.3.3 Execution of Generalized Meta Learning

In GML, optimization of each problem should be performed in a topologically reverse order, as the upper-level optimization is constrained by the result of lower-level optimization. To ease an GML implementation, we also automate such an execution order with the dataflow graph developed in Section 2.3.1. Specifically, let's assume that there is a lower-to-upper edge between problems P_i and P_j (*i.e.* $\theta_i^* \rightarrow \theta_j^*$). When the optimization process (*i.e.* a small number of gradient steps) of the problem P_i is complete, it can call the problem P_j to start its one-step gradient descent update through the lower-to-upper edge. The problem P_j waits until all lower level problems in $\mathscr{L}_j$ send their calls, and then performs the one-step gradient descent update when all the calls from lower levels are received. Hence, to achieve the full execution of gradient-based GML, we only need to call the one-step gradient descent processes of the lowermost problems, as the optimization processes of upper problems will be automatically called from lower problems via lower-to-upper edges.

To summarize, automatic differentiation for GML is accomplished by performing gradient updates of multiple optimization problems in a topologically reverse order based on the lower-to-upper edges (Sec. 2.3.3), where gradients for each problem are calculated by iteratively multiplying best-response Jacobians obtained with ITD/AID (Sec. 2.3.2) while reverse-traversing the dataflow graph (Sec. 2.3.1).

2.4 Software Design

On top of the automatic differentiation technique developed in Section 2.3, we build an easy-to-use and modular software library, BETTY, with various systems support for large-scale gradient-based GML. In detail, we break down GML into two high-level concepts, namely 1) optimization problems and 2) hierarchical dependencies among problems, and design abstract Python classes for both of them. Such abstraction is also motivated by our dataflow graph interpretation, as each of these concepts respectively corresponds to nodes and edges. The architecture of BETTY is shown in Figure 2.1.

Problem Each optimization problem P_k in GML is defined by the parameter (or module) θ_k, the sets of the upper and lower constraining problems $\mathscr{U}_k$ & $\mathscr{L}_k$, the dataset $\mathscr{D}_k$, the cost function $\mathscr{C}_k$, the optimizer, and other optimization configurations (*e.g* best-response Jacobian calculation algorithm, number of unrolling steps). The `Problem` class is an interface where users can provide each of the aforementioned components to define the optimization problem. In detail, each one except for the cost function $\mathscr{C}_k$ and the constraining problems $\mathscr{U}_k$ & $\mathscr{L}_k$ can be provided through the class constructor, while the cost function can be defined through a *"training_step"* method and the constraining problems are automatically provided by `Engine`.

Abstracting an optimization problem by encapsulating module, optimizer, and data loader together additionally allows us to implement various systems support, including mixed-precision, data-parallel training, and gradient accumulation, within the abstract `Problem` class. A similar strategy has also been adopted in popular frameworks for large-scale deep learning such as DeepSpeed [129]. Since implementations of such systems support as well as best-response Jacobian are abstracted away, users can easily plug-and-play different algorithmic and systems design choices, such as unrolling steps or mixed-precision training, via `Config` in a modular fashion. We summarize the supported features within BETTY In Table 2.1, and provide an example usage of `Problem` in Listing 2.1.

Category	Features
Best-response Jacobian algorithms	· ITD-RMAD · AID-FD · AID-NMN · AID-CG
Systems	· Mixed-precision · Data-parallel · Gradient accumulation
Logging	· Default Python logging · TensorBoard · Weights & Biases
Miscellaneous	· Gradient clipping · Early stopping

Table 2.1 Supported features in BETTY

```
class MyProblem(Problem):
    def training_step(self, batch):
        # Users define the cost function here
        return cost_fn(batch, self.module, self.other_probs, ...)
config = Config(type="darts", unroll_steps=10, fp16=True, gradient_accumulation=4)
prob = MyProblem("myproblem", config, module, optimizer, data_loader)
```

Listing 2.1 `Problem` class example.

Engine While `Problem` manages each optimization problem, `Engine` handles hierarchical dependencies among problems in the dataflow graph. As discussed in Section 2.3.1, a dataflow graph for GML has upper-to-lower and lower-to-upper directed edges. We allow users to define two separate graphs, one for each type of edge, using a Python dictionary, in which keys/values respectively represent start/end nodes of the edge. When user-defined dependency graphs are provided, `Engine` compiles them and finds all paths required for automatic differentiation with a modified depth-first search algorithm. Moreover, `Engine` sets constraining problem sets for each problem based on the dependency graphs, as mentioned

above. Once all initialization processes are done, users can run a full GML program by calling `Engine`'s run method, which repeatedly calls the one-step gradient descent procedure of lowermost problems. The example usage of `Engine` is provided in Listing 2.2.

```
prob1 = MyProblem1(...)
prob2 = MyProblem2(...)
dependency = {"u2l": {prob1: [prob2]}, "l2u": {prob1: [prob2]}}
engine = Engine(problems=[prob1, prob2], dependencies=dependency)
engine.run()
```

Listing 2.2 `Engine` class example.

2.5 Experiments

To showcase the general applicability of BETTY, we implement four GML benchmarks with varying complexities and scales: differentiable neural architecture search (Sec. 2.5.1), data reweighting for class imbalance (Sec. 2.5.2), correcting and reweighting corrupted labels (Sec. 2.5.3), and domain adaptation for a pretraining/finetuning framework (Sec. 2.5.4). Dataflow graphs for these experiments are provided in Figure 2.3. Furthermore, we investigate the effect of different best-response Jacobian algorithms and system features by reporting GPU memory usage and training wall time, and include more extensive analyses in Sec 2.5.5. Last but not least, in the Appendix, we include code examples (Appendix A.1), and additional training details such as hyperparameters (Appendix A.2).

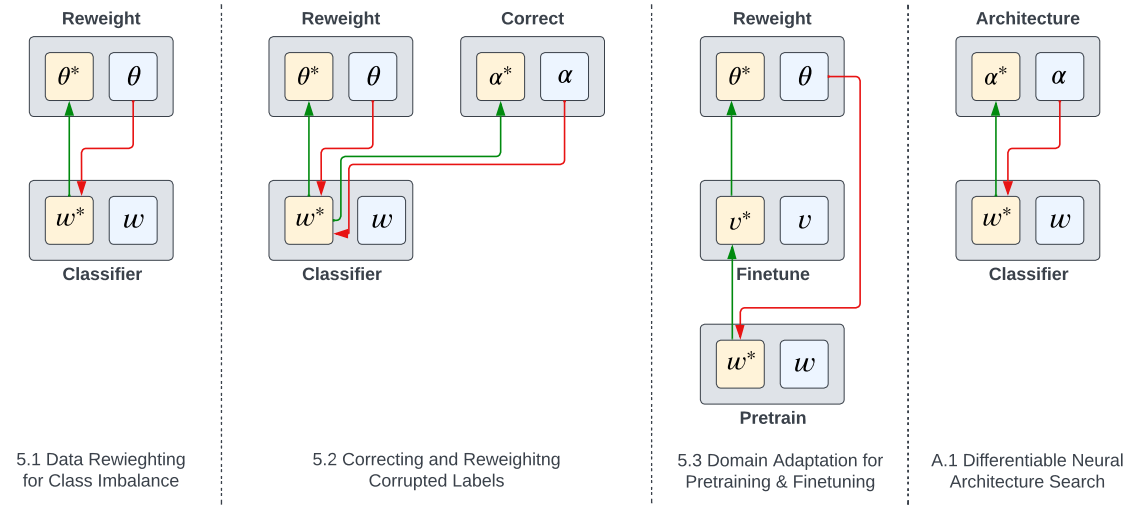


Fig. 2.3 Dataflow graphs for all our experiments

2.5.1 Differentiable Neural Architecture Search

A neural network architecture plays a significant role in deep learning research. However, the search space of neural architectures is so large that manual search is almost impossible.

To overcome this issue, DARTS [99] proposes an efficient gradient-based neural architecture search method based on the bilevel optimization formulation:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}}\ \mathscr{L}_{val}(w^*(\alpha), \alpha) \qquad \triangleright \text{ Architecture Search}$$
$$\text{s.t. } w^*(\alpha) = \underset{w}{\operatorname{argmin}}\ \mathscr{L}_{train}(w; \alpha) \qquad \triangleright \text{ Classification}$$

where α is the architecture weight and w is the network weight. The original paper uses implicit differentiation with finite difference as its best-response Jacobian algorithm to solve the above MLO program.

We follow the training configurations from the original paper's CIFAR-10 experiment, with a few minor changes. While the original paper performs a finite difference method on the initial network weights, we perform it on the unrolled network weights. This is because we view their best-response Jacobian calculation from the implicit differentiation perspective, where the second-order derivative is calculated based on the unrolled weight. This allows us to unroll the lower-level optimization for more than one step as opposed to strict one-step unrolled gradient descent of the original paper. A similar idea was also proposed in iDARTS [178]. Specifically, we re-implement DARTS with implicit differentiation and finite difference using 1 and 3 unrolling steps. The results are provided in Table 2.2.

	Algorithm	Test Acc.	Parameters	Memory	Wall Time
Random Search	Random	96.71%	**3.2M**	N/A	N/A
DARTS (original)	AID-FD*	97.24%	3.3M	10493MiB	25.4h
DARTS (ours, step=1)	AID-FD	**97.39%**	3.8M	**10485MiB**	**23.6h**
DARTS (ours, step=3)	AID-FD	97.22%	**3.2M**	**10485MiB**	28.5h

Table 2.2 DARTS re-implementation results. AID-FD refers to implicit differentiation with a finite difference method, and * indicates the difference in the implementation of AID-FD explained above.

Our re-implementation with different unrolling steps achieves a similar performance as the original paper. We also notice that our re-implementation achieves slightly less GPU memory usage and wall time. This is because the original implementation calculates gradients for the architecture weights (upper-level parameters) while running lower-level optimization, while ours only calculates gradients of the parameters for the corresponding optimization stage.

2.5.2 Data Reweighting for Class Imbalance

Many real-world datasets suffer from class imbalance due to underlying long-tailed data distributions. Meta-Weight-Net (MWN) [145] proposes to alleviate the class imbalance issue with a data reweighting scheme where they learn to assign higher/lower weights to data

from more rare/common classes. In detail, MWN formulates data reweighting with bilevel optimization as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ \mathscr{L}_{val}(w^*(\theta)) \qquad \triangleright \text{ Reweighting}$$

$$\text{s.t. } w^*(\theta) = \underset{w}{\operatorname{argmin}}\ \frac{1}{N}\sum_{i=1}^{n} \mathscr{R}(L^i_{train};\theta)\cdot L^i_{train}(f(x_i;w),y_i) \qquad \triangleright \text{ Classification}$$

where w is the network parameters, L^i_{train} is the training loss for the i-th training sample, and θ is the MWN $\mathscr{R}$'s parameters, which reweights each training sample given its training loss L^i_{train}.

Following the original paper, we artificially inject class imbalance into the CIFAR-10 dataset by geometrically decreasing the number of data sample for each class, as per an imbalance factor. While the official implementation, which is built upon Torchmeta [32], only adopts ITD-RMAD for best-response Jacobian calculation, we re-implement MWN with multiple best-response Jacobian algorithms, which only require one-liner changes using BETTY, to study their effect on test accuracy, memory efficiency, and training wall time. The experiment results are given in Table 2.3.

	Algorithm	IF 200	IF 100	IF 50	Memory	Time
MWN (original)	ITD-RMAD	68.91	75.21	80.06	2381MiB	35.8m
MWN (ours, step=1)	ITD-RMAD	71.96	75.13	79.50	2381MiB	36.0m
MWN (ours, step=1)	AID-CG	66.23±1.88	70.88±1.68	75.41±0.61	2435MiB	67.4m
MWN (ours, step=1)	AID-NMN	66.45±1.18	70.92±1.35	75.90 ±1.73	2419MiB	67.1m
MWN (ours, step=1)	AID-FD	75.45±0.63	78.11±0.43	81.15±0.25	**2051MiB**	**28.5m**
MWN (ours, step=5)	AID-FD	**76.56**±1.19	**80.45**±0.73	**83.11**±0.54	**2051MiB**	65.5m

Table 2.3 MWN experiment results. IF denotes an imbalance factor. AID-CG/NMN/FD respectively stand for implicit differentiation with conjugate gradient/Neumann series/finite difference.

We observe that different best-Jacobian algorithms lead to vastly different test accuracy, memory efficiency, and training wall time. Interestingly, we notice that AID-FD with unrolling steps of both 1 and 5 consistently achieve better test accuracy (close to SoTA [154]) and memory efficiency than other methods. This demonstrates that, while BETTY is developed to support large and general GML programs, it is still useful for simpler bilevel optimization tasks as well. An additional analysis on the effect of best-response Jacobian can also be found in Appendix 2.5.5.

Furthermore, to demonstrate the scalability of BETTY to large-scale GML, we applied MWN to sentence classification with the BERT-base model [35] with 110M parameters. Similarly, we artificially inject class imbalance into the SST dataset, and use AID-FD as our best-response Jacobian calculation algorithm. The experiment results are provided in Table 2.4.

As shown above, default full-precision training fails due to the CUDA out-of-memory error, while mixed-precision training, which only requires a one-line change in `Config`,

	Algorithm	IF 20	IF 50	Memory
Baseline	AID-FD	89.99±0.38	87.54±0.70	8319MiB
MWN (fp32)	AID-FD	-	-	Out-of-memory
MWN (fp16)	AID-FD	**91.06**±0.09	**89.79**±0.65	10511MiB

Table 2.4 MWN+BERT experiment results. fp32 and fp16 respectively stand for full-precision and mixed-precision training.

avoids this issue while also providing consistent improvements in test accuracy compared to the BERT baseline. This demonstrates that our system features are indeed effective in scaling GML to large models. We include more analyses on our systems support in Appendix A.3.

2.5.3 Correcting & Reweighting Corrupted Labels

Another common pathology in real-world data science is the issue of label corruption, stemming from noisy data preparation processes (*e.g.* Amazon MTurk). One prominent example of this is in weak supervision [134], where users create labels for large training sets by leveraging multiple weak/noisy labeling sources such as heuristics and knowledge bases. Due to the nature of weak supervision, generated labels are generally noisy, and consequently lead to a significant performance degradation. In this example, we aim to mitigate this issue by 1) correcting and 2) reweighting potentially corrupted labels. More concretely, this problem can be formulated as an *extended* bilevel optimization problem, as, unlike the MWN example, we have two optimization problems—correcting and reweighting—in the upper level, as opposed to one. The mathematical formulation of this GML program is as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathscr{L}_{val}(w^*(\theta,\alpha)), \quad \alpha^* = \underset{\alpha}{\operatorname{argmin}}\, \mathscr{L}'_{val}(w^*(\theta,\alpha)) \qquad \triangleright \text{ RWT \& CRT}$$

$$\text{s.t. } w^*(\theta,\alpha) = \underset{w}{\operatorname{argmin}}\, \frac{1}{N}\sum_{i=1}^{n} \mathscr{R}(L^i_{train};\theta)\cdot L^i_{train}(f(x_i;w), g(x_i,y_i;\alpha)) \qquad \triangleright \text{ Classification}$$

where, α is the parameter for the label correction network g, and $\mathscr{L}'_{val}$ is augmented with the classification loss of the correction network in addition to that of the main classification network f on the clean validation set.

We test our framework on two noisy label benchmarks, namely WRENCH and Clothing-1M, to demonstrate the general applicability and effectiveness of BETTY.

WRENCH WRENCH benchmark [177] contains multiple weak supervision datasets. On each dataset, we try both 2-layer MLP and BERT-base models as our classifiers, AID-FD as our best-response Jacobian algorithm, and Snorkel Data Programming [134] and majority voting as our weak supervision algorithms for generating training labels. The experiment results are provided below.

	TREC	AGNews	IMDB	SemEval	ChemProt	YouTube
Snorkel	57.52±0.18	62.00±0.07	71.03±0.55	71.00±0.00	51.54±0.41	77.44±0.22
Baseline	53.88±1.83	80.74±0.20	72.26±0.81	71.50±0.44	54.47±0.78	88.16±1.56
+Reweight	57.56±1.41	82.79±0.10	77.18±0.13	77.23±3.38	65.33±0.72	**91.60**±0.75
+Reweight & Correct	**66.76**±1.31	**83.16**±0.20	**77.80**±0.26	**84.34**±1.43	**67.69**±1.17	**91.52**±0.66

Table 2.5 Wrench + MLP results.

	TREC	AGNews	IMDB	SemEval	ChemProt	YouTube
Baseline	64.14±6.56	86.12±0.17	71.66±2.05	79.93±1.53	52.35±0.56	93.20±1.44
+Reweight	84.07±4.42	89.62±0.60	**87.85**±0.24	87.45±0.69	71.42±1.50	94.67±0.46
+Reweight & Correct	**93.07**±0.31	**90.40**±0.16	87.45±0.39	**87.92**±0.04	**75.27**±1.23	**94.80**±0.80

Table 2.6 Wrench + BERT-base results.

Clothing-1M Clothing-1M [171] is a real-world noisy dataset that consists of 1 million fashion images collected from various online shopping websites and has the approximate noise ratio of 38.5%. Following the standard, we use ResNet-50 as our backbone model and attempt to correct and reweight noisy labels with extended bilevel optimization. The experiment result is presented in Table 2.7

	Test Accuracy
Baseline	70.76%
+Reweight	75.57%
+Reweight & Correct	**76.34**%

Table 2.7 Clothing-1M + ResNet-50 results.

We observe that simultaneously applying label correction and reweighting significantly improves the test accuracy over the baseline and the reweighting-only scheme in almost all tasks. Thanks to BETTY, adding label correction in the upper-level on top of the existing reweighting scheme only requires defining one more `Problem` class, and accordingly updating the problem dependency in `Engine` (code examples can be found in Appendix A.1).

2.5.4 Domain Adaptation for Pretraining & Finetuning

Pretraining/finetuning paradigms are increasingly adopted with recent advances in self-supervised learning [35, 68]. However, the data for pretraining are oftentimes from a different distribution than the data for finetuning, which could potentially cause negative transfer. Thus, domain adaptation emerges as a natural solution to mitigate this issue. As a domain adaptation strategy, [128] proposes to combine data reweighting with a pretraining/finetuning framework to automatically decrease/increase the weight of pretraining samples that cause negative/positive transfer. In contrast with the above two benchmarks, this problem can be

formulated as trilevel optimization as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathscr{L}_{FT}(v^*(w^*(\theta))) \qquad \triangleright \text{ Reweighting}$$

$$\text{s.t. } v^*(w^*(\theta)) = \underset{v}{\operatorname{argmin}} \left(\mathscr{L}_{FT}(v) + \lambda \|v - w^*(\theta)\|_2^2 \right) \qquad \triangleright \text{ Finetuning}$$

$$w^*(\theta) = \underset{w}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{n} \mathscr{R}(x_i; \theta) \cdot L_{PT}^i(w) \qquad \triangleright \text{ Pretraining}$$

where x_i / L_{PT}^i stands for the i-th pretraining sample/loss, $\mathscr{R}$ for networks that reweight importance for each pretraining sample x_i, and λ for the proximal regularization parameter. Additionally, w, v, and θ are respectively parameters for pretraining, finetuning, and reweighting networks.

We conduct an experiment on the OfficeHome dataset [159] that consists of 15,500 images from 65 classes and 4 domains: Art (Ar), Clipart (Cl), Product (Pr), and Real World (RW). Specifically, we randomly choose 2 domains and use one of them as a pretraining task and the other as a finetuning task. ResNet-18 [69] is used for all pretraining/finetuning/reweighting networks, and AID-FT with an unrolling step of 1 is used as our best-response Jacobian algorithm. Following [10], the finetuning and the reweighting stages share the same training dataset. We adopted a normal pretraining/finetuning framework without the reweighting stage as our baseline, and the result is presented in Table 2.8.

	Algorithm	Cl→Ar	Ar→Pr	Pr→Rw	Rw→Cl	Memory	Time
Baseline	N/A	65.43±0.36	87.62±0.33	77.43±0.41	68.76±0.13	**3.8GiB**	**290s**
+ Reweight	AID-FD	**67.76**±0.83	**88.53**±0.42	**78.58**±0.17	**69.75**±0.43	8.2GiB	869s

Table 2.8 Domain Adaptation for Pretraining & Finetuning results. Reported numbers are classification accuracy on the target domain (right of arrow), after pretraining on the source domain (left of arrow). We note that *Baseline* is a two-layer, and *Baseline + Reweight* a three-layer, GML program.

Our trilevel optimization framework achieves consistent improvements over the baseline for every task combination at the cost of additional memory usage and wall time, which demonstrates the empirical usefulness of generalized meta learning beyond a two-level hierarchy. Finally, we provide an example of (a simplified version of) the code for this experiment in Appendix A.1 to showcase the usability of our library for a general GML program.

2.5.5 Design Choice Analysis

In this section, we visually compare the convergence speed of different best-response Jacobian algorithms with the loss convergence graphs on the synthetic hyperparameter optimization task and the data reweighting task (Section 2.5). Specifically, we analyze the convergence

speed in terms of both 1) the number of steps and 2) training time, as the per-step computational cost differs for each algorithm.

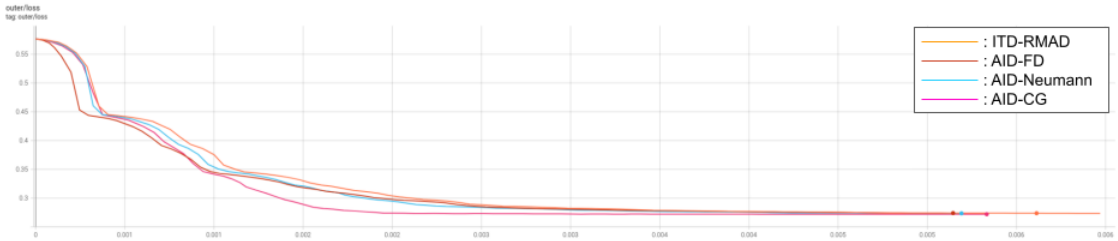
Synthetic Hyperparameter Optimization

Following [53], we constructed a synthetic hyperparameter optimization task where we optimize the weight decay value for *every* parameter in simple binary logistic regression. Mathematically, this problem can be formulated as bilevel optimization as follows:

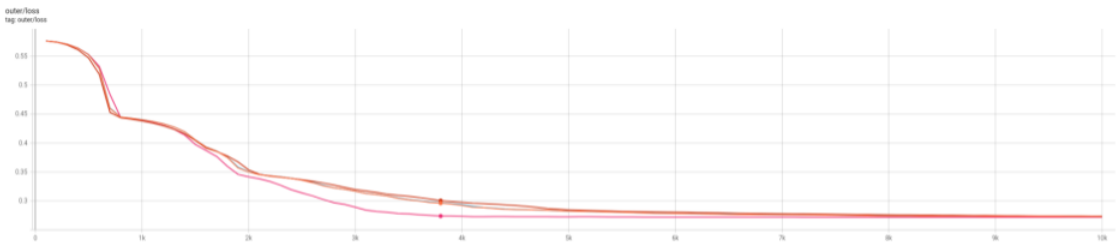
$$\lambda^* = \arg\min_{\lambda} \text{ sigmoid}(y_u x_u^T w^*)$$
$$w^* = \arg\min_{w} \text{ sigmoid}(y_l x_l^T w^*) + \frac{1}{2} w^T diag(\lambda) w$$

where, (x_l, y_l) and (x_u, y_u) are repsectively the training datasets for the lower-(and upper-)level problems, with $x \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^{n \times 1}$. Here, n is the number of training data in each dataset and d is the dimension of the feature vector. $w \in \mathbb{R}^{d \times 1}$ is the logistic regression parameter, and $\lambda \in \mathbb{R}^{d \times 1}$ is the hyperparameter (i.e. the per-parameter weight decay value).

Given the above setup, we compared four different best-reponse Jacobian algorithms: 1) ITD-RMAD, 2) AID-FD, 3) AID-CG, and 4) AID-Neumann. For the fair comparison, we fixed the unrolling step to 100 for all algorithms. The experiment result is presented below:



(a) Outer training loss (x-axis: time)



(b) Outer training loss (x-axis: step)

Fig. 2.4 Convergence analysis of different best-response Jacobian algorithms on the synthetic hyperparameter optimization task

As shown in Figure 2.4, AID-CG achieves the fastest convergence both in terms of training steps and training time. However, AID-FD achieves the fastest per-step computation

time as it is the only algorithms that doesn't require the explicit calculation of the second-order derivative (i.e. Hessian).

Data Reweighting

To study how different best-response Jacobian algorithms perform on more complex tasks, we repeated the above experiment on the data reweighting task from Section 5.1. Again, for the fair comparison, we used the same unrolling step of 1 for all algorithms. The experiment result is provided in Figure 2.5.

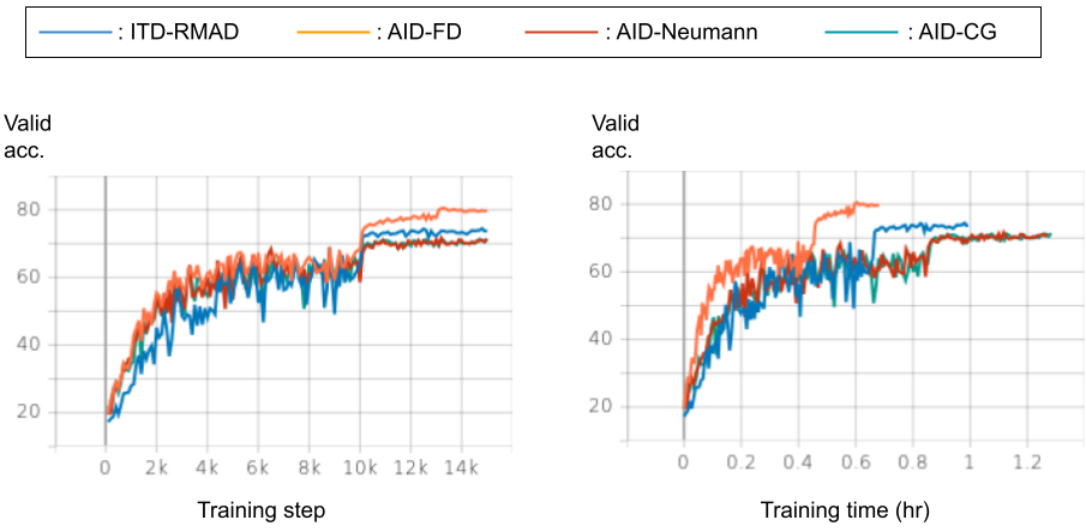


Fig. 2.5 Convergence analysis of different best-response Jacobian algorithms on the data reweighting task

.

Unlike in the synthetic hyperparameter optimization task, AID-FD achieves the fastest convergence in terms of training steps and training time as well as the best final validation accuracy. As AID-FD doesn't require any second-order derivative calculation, it also achieves the minimal per-step computation cost.

Above two experiments follow the no free lunch theorem: the optimal design choice can vary for different tasks without golden rules. However, thanks to the modular interface for switching between different design choices (in `Config`), only minimal programming efforts would be needed with BETTY, expediting the research cycle.

2.6 Related Work

Bilevel & Generalized Meta Learning There are a myriad of machine learning applications that are built upon bilevel optimization (BLO), the simplest case of generalized

meta learning with a two-level hierarchy. For example, neural architecture search [99, 178], hyperparameter optimization [45, 103, 105], reinforcement learning [72, 89], data valuation [137, 164], meta learning [43, 130], and label correction [181] are formulated as BLO. In addition to applying BLO to machine learning tasks, a variety of optimization techniques [30, 53, 80, 100] have been developed for solving BLO.

Following the popularity of BLO, GML with more than a two-level hierarchy has also attracted increasing attention recently [128, 148, 152, 172]. In general, these works construct complex multi-stage ML pipelines, and optimize the pipelines in an end-to-end fashion with GML. For instance, [50] constructs the pipeline of (data generation)–(architecture search)–(classification) and [70] of (data reweighting)–(finetuning)–(pretraining), all of which are solved with GML. Furthermore, [141] study gradient-based methods for solving GML with theoretical guarantees.

Generalized Meta Learning Software There are several software libraries that are frequently used for implementing GML programs. Most notably, *JAXopt* [17] proposes an efficient and modular approach for AID by leveraging JAX's native autodiff of the optimality conditions. Despite its easy-to-use programming interface for AID, it fails to support combining the chain rule with AID as in Equation (2.2), because it overrides the default behavior of JAX's automatic differentiation, which takes care of the chain rule. Therefore, it cannot be used for implementing GML beyond a two-level hierarchy without major changes in the source code and the software design. Alternatively, *higher* [54] provides two major primitives of making 1) stateful PyTorch modules stateless and 2) PyTorch optimizers differentiable to ease the implementation of AID/ITD. However, users still need to manually implement complicated internal mechanisms of these algorithms as well as the chain rule with the provided primitives. *Torchmeta* [32] also provides similar functionalities as *higher*, but it requires users to use its own stateless modules implemented in the library rather than patching general modules as in *higher*. Thus, it lacks the support for user's custom modules, limiting its applicability. *learn2learn* [5] focuses on supporting meta learning. However, since meta-learning is strictly a bilevel problem, extending it beyond a two-level hierarchy is not straightforward. Finally, most existing libraries do not have systems support, such as data-parallel training, that could mitigate memory/compute bottlenecks.

2.7 Conclusion

In this work, we aimed to help establish both mathematical and systems foundations for automatic differentiation in GML. To this end, we devised a novel dataflow graph for GML, upon which an automatic differentiation procedure is built, and additionally introduced BETTY, a software library with various systems support, that allows for easy programming of a wide range of GML applications in a modular fashion. We showed that BETTY allows for scaling up to both larger models with many parameters, as well as to GML programs with multiple dependent problems.

Chapter 3

Scaling Meta Learning

Despite many benefits BETTY brings in, the framework itself doesn't provide any algorithmic innovations that can further improve scalability and stability in meta learning, both of which are crucial in practical automated ML. In this chapter, we develop SAMA, a variant of the implicit differentiation that demonstrates significantly improved compute/memory efficiency, and additionally introduce communication optimization in the distributed training setting for the improved scalability.

3.1 Introduction

Meta learning aims to learn the *inductive biases* (*e.g.* training data, neural architecture) of a machine learning program in such a way that a model trained with these inductive biases achieves optimal performance on user-specified *objectives* (*e.g.* fairness, quick generalization). This concept of meta learning can naturally be formulated as bilevel optimization, where the upper (meta) level problem encodes inductive biases and objectives, and the lower (base) level optimization problem represents the main machine learning program of interest, such as image classification or language modeling. Depending on the design of inductive biases and objectives, meta learning has found many applications in machine learning, including hyperparameter optimization [46], data optimization [58, 145], neural architecture search [99, 178], learned optimizers [109, 110], and few-shot learning [43, 130].

Following its versatility, numerous algorithms have been proposed to solve meta learning. Among them, gradient-based meta learning (GBML) has in particular gained considerable attention, due to its capability to optimize a wide range of *high-dimensional* inductive biases in an *efficient* manner. For example, MAML [43] finds optimal initialization weights (inductive bias) that achieve quick generalization to new tasks (objective), and L2RW [136] optimizes training sample weights (inductive bias) to achieve robustness against label noise (objective). However, the above benefits of GBML oftentimes get overshadowed by its poor scalability in practice, especially under the recent trend of large models [19, 86, 125], which arises due to several factors. First, many GBML algorithms [43, 103, 130] require inversion of a large Jacobian matrix, which suffers from both algorithmic instability as well as exorbitant compute/memory costs. Second, most GBML research assumes that lower

	Constant Memory	Jacobian Inverse Free	Adaptive Optimizer Support	(Efficient) Distributed Training Support	**Overall Scalability**
Iterative Differentiation [43, 45, 105]	✗	✗	✔	✗	✗
Recurrent Backpropagation [95]	✓	✗	✔	✗	✗
$T_1 - T_2$ [99, 104]	✔	✔	✗	✗	✓
Neumann Series [103]	✔	✗	✓	✗	✓
Conjugate Gradient [130]	✔	✗	✓	✗	✓
SAMA (ours)	✔	✔	✔	✔	✔

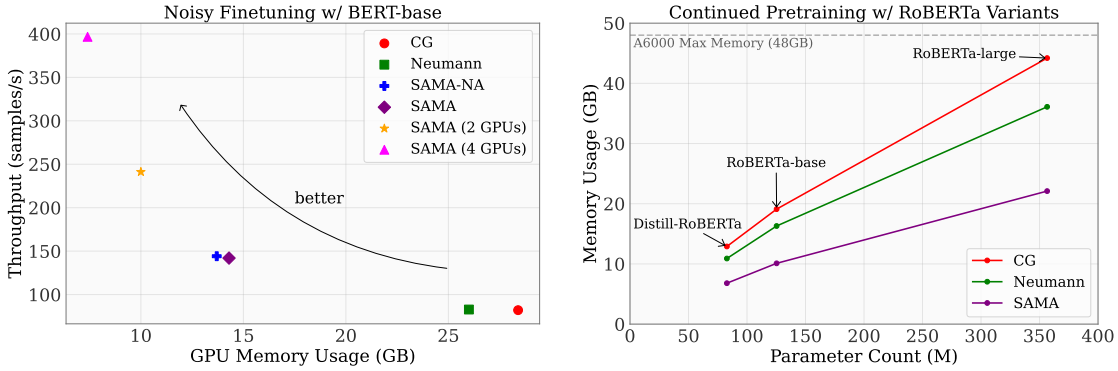


Fig. 3.1 **Top:** Table showing a scalability comparison. **Bottom left:** Plot of throughput vs memory of different GBML algorithms on the *noisy finetuning of BERT-base* experiment. SAMA achieves better memory/compute efficiency overall given a fixed model, and the gap further widens by distributing compute across multiple GPUs with our efficient distributed training strategy. **Bottom right:** Plot of memory vs model size (*i.e.*, # of parameters) of different GBML algorithms on the *continued pretraining of RoBERTa* experiment. SAMA demonstrates the least significant increase in GPU memory usage with the increasing model size compared to baseline methods.

(base) level optimization is performed with SGD, whereas most large models, exemplified by Transformers [158], are by default optimized with adaptive optimizers like Adam [85]; consequently, the applicability of SGD-based GBML methods to large models trained with adaptive optimizers remains unclear. Finally, most GBML research to date has been limited to the single-GPU setup due to the lack of distributed training support [32, 54], which is essential in large-scale learning.

In this work, we endeavor to resolve the aforementioned scalability issues in GBML by co-developing algorithms and systems, and explore the initial potential of scalable meta learning in diverse applications. Our main contributions can be summarized as follows:

1. We investigate the root causes of substantial memory/compute costs, algorithmic instability, and a lack of distributed training support in GBML, all of which significantly limit its scalability, through a technical analysis of implicit differentiation. In doing so, we identify three major factors, namely (i) base Jacobian inversion, (ii) a lack of algorithmic adaptation for adaptive optimizers, and (iii) a need for the custom implementation of the backward pass of meta gradient computation, and discuss how each of them contributes negatively to the above limitations in depth.

2. Taking one step further, we propose an initial solution to each of the aforementioned issues by respectively (i) approximating the base Jacobian with an identity matrix, (ii) additionally expanding the meta Jacobian via the chain rule, and (iii) devising a novel communication strategy that efficiently uses the communication-computation overlap trick [93]. Combining all these solutions, we develop SAMA, a holistic and practically **S**cal**A**ble **M**eta learning **A**lgorithm.

3. We evaluate the scalability and the overall performance of SAMA on a multitude of large-scale meta learning benchmarks involving large language models (*e.g.*, BERT [35] and RoBERTa [101]) or large datasets (*e.g.*, ImageNet-1k [33]). Notably, SAMA showcases up to 1.7/4.8× increase in throughput and 2.0/3.8× decrease in memory consumption respectively on single-/multi-GPU setups compared to other baseline meta learning algorithms. In addition, we observe that SAMA-based data optimization consistently leads to improvements in text classification accuracy with large language models, and achieves state-of-the-art results in both small-/large-scale data pruning, demonstrating the initial potential of scalable meta learning.

3.2 Background

We begin by reviewing the basics of (gradient-based) meta learning in order to establish the key aspects that have limited its scalability. Mathematically, meta learning is commonly formulated as bilevel optimization as follows:

$$\begin{aligned} \lambda^* &= \underset{\lambda}{\operatorname{argmin}}\, L_{meta}(D_{meta}; \theta^*(\lambda)) \\ s.t.\ \ \theta^*(\lambda) &= \underset{\theta}{\operatorname{argmin}}\, L_{base}(D_{base}; \theta, \lambda) \end{aligned}$$

where λ (respectively, θ) are the parameters of meta (base) learners, D_{meta} (D_{base}) are meta (base) datasets, and L_{meta} (L_{base}) are meta (base) loss functions. An important implication of the above formulation is that meta learning changes the task of finding the optimal inductive biases from designing heuristics to designing meta optimization problems. As an example, consider the problem of finding the optimal inductive bias for fair classification given class-imbalanced training data. A traditional approach to this problem is to use a heuristic that reweights training samples inversely proportional to class frequencies. On the contrary, L2RW [136] designs a meta optimization problem by curating a small number of class-balanced data for the meta dataset D_{meta} and setting the meta learner λ to be importance weights for all training data. In short, unlike heuristic-based methods that explicitly specify "how to learn," meta learning methods only specify "what to learn" and let the meta learner automatically determine "how." From a programming paradigm perspective, such a difference can be understood as a transition from imperative to declarative programming.

While there are multiple approaches to solving meta learning, in this work we focus on *gradient-based* approaches due to their ability to efficiently solve high-dimensional meta optimization (*i.e.* $\dim(\lambda) \gg 1$) problems. Such an ability is essential given that the

search space for inductive biases (*e.g.* importance weights for training data) can increase exponentially with recent large models and datasets. Concretely, GBML computes a meta gradient composed of two terms—the best-response Jacobian and direct gradient—with the chain rule, as follows:

$$\frac{\partial L_{meta}}{\partial \lambda} = \underbrace{\frac{\partial \theta^*}{\partial \lambda}}_{\text{best-response Jacobian}} \cdot \underbrace{\frac{\partial L_{meta}}{\partial \theta^*}}_{\text{direct gradient}} \tag{3.1}$$

Since the direct gradient (teal) computation is straightforward with the underlying automatic differentiation library, the major challenge in GBML lies in computing the best-response Jacobian (purple), of which two common solutions are iterative differentiation [43, 45, 46, 105] and implicit differentiation [66, 103, 122, 130]. Between these two, in this paper we adopt implicit differentiation as our baseline solution to GBML, as it achieves better computation and memory efficiency than iterative differentiation [53], both of which are vital in accomplishing our goal of scalable GBML.

The gist of implicit differentiation is that it calculates the best-response Jacobian by leveraging Cauchy's Implicit Function Theorem (IFT) and re-interpreting the base optimization problem from the perspective of fixed-point iteration given an iterative solver u, as follows:

$$\frac{\partial \theta^*}{\partial \lambda} = -\underbrace{\frac{\partial u}{\partial \lambda}}_{\text{meta Jacobian}} \cdot \left(\underbrace{\frac{\partial u}{\partial \theta^*}}_{\text{base Jacobian}} \right)^{-1} \quad \text{where} \begin{cases} \theta^* = \lim_{t \to \infty} \theta_t \\ \theta_t = \theta_{t-1} - u(\theta_{t-1}; \lambda) \end{cases} \tag{3.2}$$

While exact implicit differentiation requires solving the base optimization problem to convergence θ^* by repeatedly applying an iterative solver u (*e.g.*, SGD or Adam) to calculate base (blue) and meta (red) Jacobians, this is computationally impractical, especially in most large-scale learning settings. Therefore, researchers oftentimes approximate θ^* with a small number of unrolled update steps of u. This results in a solution that alternates gradient descent between base and meta optimization problems, where the base gradient is calculated with standard backpropagation and the meta gradient with Eqs. (3.1) & (3.2). Noting that many techniques have been developed to efficiently perform and scale up standard backpropagation, we deduce that the major challenges in scaling GBML lie in meta gradient computation, which will be discussed in depth in the next section.

3.3 Scaling Meta Learning

It has long been recognized that meta gradient computation in GBML suffers from a substantial compute/memory cost [103, 130], algorithmic instability [4, 36], and a lack of efficient distributed training support [5, 24, 32], all of which can significantly limit its scalability. In this section, we first attempt to understand the above limitations at a technical level. Toward this end, we investigate three aspects in Eqs. (3.1) & (3.2), namely (i) base Jacobian inversion, (ii) algorithmic adaptation for adaptive optimizers, and (iii) a need for the custom imple-

mentation of meta gradient computation, and discuss how they lead to the aforementioned limitations. Next, we propose initial solutions for each of these issues, based on which we build a holistic and **S**cal**A**ble **M**eta Learning **A**lgorithm, SAMA.

3.3.1 Base Jacobian Inverse

Problem Denoting the size of the base learner (*i.e.*, $\dim(\theta)$) as n_b, the computational complexity of the naive base Jacobian (blue) inversion in Eq. (3.2) is $\mathcal{O}(n_b^3)$. Since such cubic complexity is impractical even for small-sized base learners, practitioners typically utilize various linear systems techniques such as Neumann series [103] or conjugate gradient [130], and directly *approximate* $\left(\frac{\partial u}{\partial \theta^*}\right)^{-1}\frac{\partial L_{meta}}{\partial \theta^*}$. Given that the base Jacobian is a function of the Hessian matrix of the base optimization problem in GBML, these algorithms solve linear systems by iteratively performing Hessian-vector products. However, this Hessian-vector product computation contributes negatively to all three above limitations. First, while many automatic differentiation engines provide an efficient Hessian-vector product implementation like Pearlmutter's algorithm [120], its memory/compute cost is still prohibitive with larger models, as demonstrated in Fig. 3.1. Second, in most cases, we only have access to a stochastic estimation of the Hessian due to mini-batch sampling [90]. Hence, meta gradients obtained with noisy Hessian-vector products can be biased, which may result in training instability. Finally, the most efficient distributed training features, such as communication-computation overlap [93], are designed for first-order gradients, rather than the higher-order gradients involved in Hessian-vector products.

Solution A simple solution for avoiding the aforementioned issues stemming from base Jacobian inversion is to approximate the base Jacobian with an identity matrix as follows:

$$\frac{\partial L_{meta}}{\partial \lambda} = -\frac{\partial u}{\partial \lambda}\cdot\left(\frac{\partial u}{\partial \theta^*}\right)^{-1}\cdot\frac{\partial L_{meta}}{\partial \theta^*} \approx -\frac{\partial u}{\partial \lambda}\cdot\frac{\partial L_{meta}}{\partial \theta^*} \tag{3.3}$$

Under the deep equilibrium model setting, Jacobian-free backpropagation [47] shows that such an approximation can be understood as preconditioning the original meta gradient. Our approximation also resembles approximating the Hessian as an identity matrix in one-step unrolling techniques (*i.e.* $T_1 - T_2$) from [99, 104]. While this approximation is exact when the iterative solver u is naive SGD where $u = \frac{\partial L_{base}}{\partial \theta}$, we note that the base Jacobian does not necessarily equate with the Hessian when an adaptive optimizer is used in base optimization (more detailed discussion on this issue is deferred to Sec. 3.3.2). Furthermore, their methods calculate the meta Jacobian at initialization θ instead of at convergence θ^* due to their close connection to iterative differentiation [105], and thereby inhibit unroll steps larger than 1, unlike our approach. Considering that a larger number of unroll steps allows for less frequent computations of expensive meta gradients, our implicit-differentiation-based derivation can lead to a further computational gain in large-scale meta learning.

Justification of the Identity Approximation for Base Jacobian

In this section, we aim to study the empirical effect of our identity approximation for base Jacobian on the meta gradient $\frac{\partial L_{meta}}{\partial \lambda}$ and the optimal meta solution λ^*, when the true base Jacobian is not an identity matrix, as a part of the justification of our approximation. As obtaining the closed-form solution of the Hessian is impossible in almost all deep learning problems, we study the soundness of the identity approximation of base Jacobian in the simpler "biased regression" setting [53], for which the bilevel optimization formulation is as follows:

$$\lambda^* = \arg\min_{\lambda} \|X' w^*(\lambda) - y'\|^2$$
$$w^*(\lambda) = \arg\min_{w} \|Xw - y\|^2 + \beta\|w - \lambda\|^2$$

Given the above formulation, the closed-form solutions for the base Jacobian, the meta-gradient g_λ, and the optimal meta solution λ^* are:

1. Base Jacobian $= X^T X + \beta I$
2. $g_\lambda = \beta(X^T X + \beta I)^{-1}(X'^T X' w^* - X'^T y')$, where $w^* = (X^T X + \beta I)^{-1}(X^T y + \beta\lambda)$
3. $\lambda^* = (A^T A)^{-1} A^T b$, where $A = \beta X'(X^T X + \beta I)^{-1}$, $b = y' - X'(X^T X + \beta I)^{-1} X^T y$

We set $\beta = 0.1$[1] and perform 100 meta updates, and measure 1) the cosine similarity between the ground truth g_λ and the meta gradient obtained with our approximation (i.e. g_{SAMA}), and 2) the L2 distance between the current meta parameter λ_t and the optimal solution λ^* at each time step t. For a more thorough analysis, we also compute these two metrics for other meta gradient algorithms that explicitly approximate base Jacobian inverse with conjugate gradient and Neumann series. In Figure 3.2, we provide the metric obtained from all time steps.

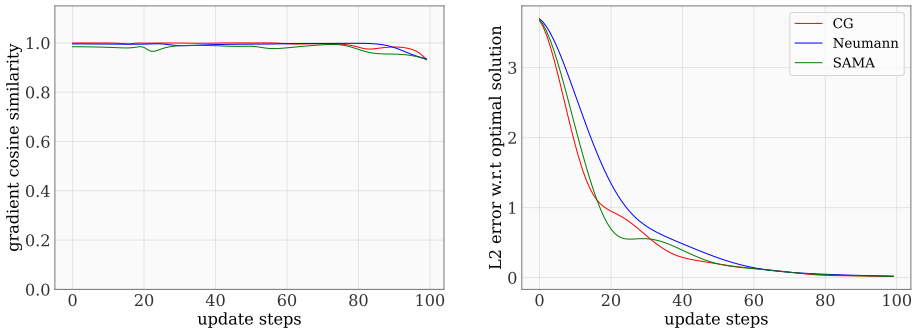


Fig. 3.2 **Left:** $\cos(g_\lambda, g_{approx})$, where g_λ is a ground truth (i.e. closed-form) meta gradient, and g_{approx} is an approximate meta gradient obtained with various popular meta learning algorithms including SAMA. **Right:** $\|\lambda_t - \lambda^*\|_2$, where λ_t is the meta parameter λ after t meta updates.

[1]We used smaller β than in the original paper [53] (1 vs 0.1), to amplify the "non-identitiness" of the base Jacobian.

From Figure 3.2, it can be clearly seen that 1) while slightly less accurate than second-order algorithms like CG, SAMA still achieves a high directional alignment with the ground truth meta-gradient, and 2) SAMA also achieves a stable convergence to the optimal solution at a comparable speed.

3.3.2 Algorithmic Adaptation for Adaptive Optimizers

Problem Most existing implicit differentiation algorithms [58, 67, 103] compute the best-response Jacobian in Eq. (3.1) based on the assumption that the iterative solver u for base optimization is vanilla SGD, whereas recent large models, exemplified by Transformers [19, 158], are by default trained with adaptive optimizers, such as Adam [85]. While the fixed point condition can be theoretically identical for any gradient-based optimizer at convergence (*i.e.*, $\frac{\partial \mathscr{L}_{base}}{\partial \theta^*} = 0$), researchers in practice approximate θ^* with a small number of gradient steps, at which the above fixed point condition would unlikely hold. Thus, the inconsistency between assumed and actual optimizers results in an incorrect meta gradient, which is a source of training instabilities and reduced performance in meta learning (as we demonstrate in Table 3.1).

Solution To take adaptive update rules into account, we propose to further expand a meta Jacobian term in Eq. (3.3) with the chain rule as follows:

$$\begin{gathered} \frac{\partial u}{\partial \lambda} = \frac{\partial g_{base}}{\partial \lambda} \cdot \frac{\partial u}{\partial g_{base}} = \frac{\partial^2 L_{base}}{\partial \lambda \partial \theta^*} \cdot \frac{\partial u}{\partial g_{base}} \\ \implies \frac{\partial L_{meta}}{\partial \lambda} \approx -\frac{\partial u}{\partial \lambda} \cdot \frac{\partial L_{meta}}{\partial \theta^*} = -\frac{\partial^2 L_{base}}{\partial \lambda \partial \theta^*} \cdot \frac{\partial u}{\partial g_{base}} \cdot \frac{\partial L_{meta}}{\partial \theta^*} \end{gathered} \tag{3.4}$$

where g_{base} is base-gradient computed at convergence (*i.e.* $\frac{\partial L_{base}}{\partial \theta^*}$). In short, we accomplish the algorithmic adaptation for any adaptive optimizer with the update rule u through the middle term $\frac{\partial u}{\partial g_{base}}$ in Eq. (3.4), which reduces to an identity matrix in the case of SGD. To analyze the adaptation cost, we note that parameter updates u in most optimizers are performed only with element-wise operations. Thus, the adaptation matrix is diagonal, for which computation/memory complexities are only $\mathscr{O}(n_b)$. As a concrete example, we provide an adaptation matrix for the most popular Adam optimizer [85] below.

$$\begin{aligned} \frac{\partial u_{adam}}{\partial g} &= \frac{\partial u}{\partial g}\left(\gamma \frac{\beta_1 m + (1-\beta_1) g}{\sqrt{\beta_1 v + (1-\beta_1) g^2} + \varepsilon}\right) \\ &= \gamma \frac{(1-\beta_1)\beta_2 v - (1-\beta_1)\beta_2 m g + (1-\beta_1)\varepsilon\sqrt{\beta_1 v + (1-\beta_1) g^2}}{\sqrt{\beta_1 v + (1-\beta_1) g^2}\left(\sqrt{\beta_1 v + (1-\beta_1) g^2} + e\right)^2} \\ &\approx \gamma \frac{(1-\beta_1)\beta_2 v - (1-\beta_1)\beta_2 m g}{\sqrt{\beta_1 v + (1-\beta_1) g^2}\left(\sqrt{\beta_1 v + (1-\beta_1) g^2} + e\right)^2} \qquad (\text{because } \varepsilon \ll 1) \end{aligned}$$

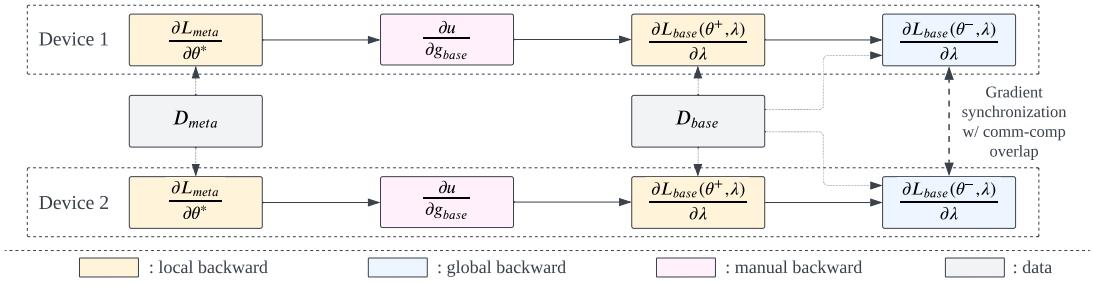


Fig. 3.3 The overall workflow of meta gradient computation with SAMA in the distributed data parallel setting. In detail, SAMA consists of three *first-order* backward passes performed with the underlying automatic differentiation engine, and one manual backward pass for algorithmic adaptation for the adaptive optimizer. Gradient synchronization is performed only once in the last backward pass with communication-computation overlap to minimize the communication bottleneck.

where the first and second moments of the gradient in Adam are denoted as m and v respectively, and the learning rate as γ.

Furthermore, to reduce computation/memory complexities involving the costly second-order derivative $\frac{\partial^2 L_{base}}{\partial \lambda \partial \theta^*}$, we instead perform the matrix-vector product with the central-difference method from DARTS [99] and the associative rule of matrix multiplication. All combined, we propose SAMA, a highly compute/memory efficient meta gradient algorithm for scalable meta learning, the formulation of which is as follows:

$$\frac{\partial L_{meta}}{\partial \lambda} \approx -\frac{\partial^2 L_{base}}{\partial \lambda \partial \theta^*} \cdot \left(\frac{\partial u}{\partial g_{base}} \cdot \frac{\partial L_{meta}}{\partial \theta^*} \right) \approx -\frac{\frac{\partial L_{base}(\theta^+,\lambda)}{\partial \lambda} - \frac{\partial L_{base}(\theta^-,\lambda)}{\partial \lambda}}{2\varepsilon} \tag{3.5}$$

where $\theta^{\pm} = \theta^* \pm \varepsilon v$ with the perturbation vector $v = \frac{\partial u}{\partial g_{base}} \cdot \frac{\partial L_{meta}}{\partial \theta^*}$ and the step size $\varepsilon = \frac{\alpha}{\|v\|_2}$. We empirically observe that $\alpha = 1.0$ works well across a multitude of tasks without further tuning. Finally, we notice that prevoius work in penalty-based bilevel optimization (*e.g.* F2SA [90] and BOME [97]) further uses the direct gradient $\frac{\partial L_{meta}}{\partial \theta^*}$ explicitly in the base level optimization to maximize the performance of the final base parameter θ^* on the meta objective. Given that our perturbation vector v includes the direct gradient term, we also follow a similar strategy and update the base parameter θ in the direction of v (*i.e.* $\theta_{t+1} = \theta_t - \varepsilon v$) every time the meta update is performed.

3.3.3 Efficient Distributed Training & Implementation

Problem In large-scale learning, distributed data parallelism (DDP), which communicates and synchronizes local gradients from each device before the parameter update, is necessary to improve both compute and memory efficiency. However, most automatic differentiation libraries like PyTorch [118] only have native DDP support for their basic backward function, whereas SAMA, similar to other meta gradient algorithms, requires a custom implemen-

tation of the backward function, which consists of three basic backward passes as shown in Eq. (3.5).[2] In addition, while a few meta learning libraries, such as Betty [24], have preliminary DDP support, they do not offer further communication cost optimization. As a result, meta learning research to date has either been limited to a single-GPU setup or suffered from communication inefficiency.

Solution Since we avoid any explicit computations of second-order gradient information in SAMA, we can utilize various efficient distributed training tricks that have been implemented for first-order gradients. More specifically, to enable efficient DDP in SAMA, we develop a novel communication strategy that performs first two backward passes locally on each device, and then overlaps computation in the final backward pass with communication. In PyTorch, this can be neatly achieved by implementing the first two backward passes with `torch.autograd.grad` and the last one with `torch.autograd.backward`. The overall workflow diagram of SAMA is presented in Figure 3.3. To facilitate research in scalable meta learning, we provide our implementation of SAMA with the above communication optimization in Betty[3] that only requires a one-line change in the configuration.

3.4 Experiments

While few-shot learning has traditionally been the most popular application of meta learning, most recent large models such as GPT [19], ViT [37], and Whisper [125], provide few-shot generalization capability out of the box. Therefore, we in this work focus on another promising application of meta learning, *data optimization*, where we transform (*e.g.*, reweight, label correct) downstream/pretraining data given the specific objectives in the meta optimization problem. Indeed, there is an increasing number of works originating in data-centric AI that empirically show that the quality of training data significantly affects the final performance of large models [1, 19, 48, 125, 143, 173]. Nevertheless, solutions proposed in these works to improve training data quality mostly rely on hand-designed heuristics, which typically result in suboptimal performance. Given that training data of large models serves as extremely high-dimensional inductive biases in machine learning, we expect that GBML's ability to efficiently optimize high-dimensional inductive biases can be fully unlocked in the data optimization application.

From a technical perspective, large-scale data optimization has a substantial compute/memory cost and frequently involves models that are trained with adaptive optimizers. Therefore, it serves as an ideal benchmark to (a) evaluate the scalability of SAMA compared to existing approaches, (b) study the effectiveness of each component in SAMA, and (c) investigate the practical usefulness of scalable meta learning across diverse domains. Specifically, in this section, we consider three data optimization applications, namely: Noisy finetuning of large language models (Sec. 3.4.1), continued pretraining of large language models (Sec. 3.4.2), and scale-agnostic data pruning (Sec. 3.4.3). While not an application

[2]Eq. (3.5) technically involves four derivatives in total, but the adaptation matrix $\frac{\partial u}{\partial g_{base}}$ can be calculated analytically without backpropagation.

[3]https://github.com/leopard-ai/betty

of data optimization, we also include a preliminary analysis on the effect of model size on few-shot image classification accuracy in Section 3.4.4. Finally, we note that all experiment details including baselines, hyperparameters, and compute resources are provided in Appendix B.2.

3.4.1 Noisy Finetuning of Large Language Models

Weak supervision [134] proposes to achieve a significant reduction in the data labeling cost by letting users quickly generate labels for a large amount of data by exploiting multiple weak labeling functions, such as hand-designed rules and other neural networks. While increasing the *quantity* of labeled data with weak supervision has led to noticeable improvements in multiple applications, a poor *quality* of generated labels results in a degradation in test accuracy, leaving room for further improvement. Here, we attempt to alleviate this data quality issue in weak supervision by utilizing meta learning to automatically optimize noisy training data guided by a small amount of clean data in the meta level. In detail, we use data reweighting [145] and label correction [180] as our data optimization operations, of which the bilevel optimization formulation is as follows:

$$\lambda^* = (\lambda_r^*, \lambda_c^*) = \underset{\lambda_r, \lambda_c}{\operatorname{argmin}} \, \mathscr{L}(D_{clean}; \theta^*(\lambda_r, \lambda_c))$$

$$\text{s.t. } \theta^*(\lambda_r, \lambda_c) = \underset{\theta}{\operatorname{argmin}} \, \frac{1}{|D_{noisy}|} \sum_{(x,y) \in D_{noisy}} w(\mathscr{L}; \lambda_r) \cdot \mathscr{L}(f(x;\theta), c(x,y;\lambda_c))$$

where $w(\cdot;\lambda_r)$ and $c(\cdot;\lambda_c)$ are meta learners, respectively, for data reweighting and label correction. To evaluate the scaling efficiency of SAMA, we perform text classification with a BERT-base model with 110M parameters on multiple weak supervision datasets from the WRENCH benchmark [177]. Furthermore, to study the effectiveness of our algorithmic adaptation strategy (Sec. 3.3.2), we conduct experiments with a variant of SAMA (*i.e.*, SAMA-NA) that does not include algorithmic adaptation, and present the experiment results in Table 3.1.

	Algorithm	TREC	SemEval	IMDB	ChemProt	AGNews	Yelp
Finetune (orig) [177]	-	66.56 (2.31)	83.93 (1.74)	79.73 (2.60)	56.09 (1.08)	86.27 (0.53)	82.26 (3.50)
COSINE [176]	-	76.56 (0.08)	86.80 (0.46)	82.98 (0.05)	58.47 (0.08)	87.03 (0.00)	89.22 (0.05)
Finetune (ours)	-	67.93 (2.55)	79.28 (1.78)	78.16 (2.28)	57.35 (1.43)	85.79 (0.49)	84.32 (2.55)
+R	SAMA-NA	74.33 (2.34)	87.11 (1.11)	81.92 (1.74)	60.88 (0.60)	86.83 (0.19)	80.96 (3.04)
+R & C	SAMA-NA	79.00 (2.62)	87.67 (1.36)	80.44 (0.97)	64.05 (0.52)	87.05 (0.39)	80.73 (3.11)
+R	SAMA	85.73 (0.81)	**89.67** (0.67)	84.31 (1.86)	76.89 (1.39)	89.05 (0.34)	93.64 (0.40)
+R & C	SAMA	**87.93** (1.17)	88.83 (2.32)	**85.71** (0.82)	**77.78** (0.59)	**89.79** (0.27)	**93.77** (0.08)

Table 3.1 WRENCH results. R and C in the first column stand for data reweighting and label correction operations. The number in parentheses indicates standard deviation for each experiment over 3 runs.

In this experiment, we are able to make two observations. First, we notice that SAMA-based data reweighting and label correction both lead to noticeable improvements over

finetuning and self-training (COSINE [176]) baselines in noisy text classification accuracy of the *large* Transformer model across all benchmarks with the help of small additional clean data D_{clean} in the meta level. Second, given the superior performance of SAMA compared to SAMA-NA, we empirically verify the importance of algorithmic adaptation when an adaptive optimizer is used to train this Transformer base learner.

Additionally, we compare compute/memory efficiency of SAMA to that of two other implicit-differentiation-based meta learning algorithms, namely Neumann Series [103] and conjugate gradient [130]. For fair comparison, we evaluate GPU memory usage (MB) and throughput (samples/sec) on the AGNews dataset from Wrench with a fixed *global* batch size of 48, and summarize the result in Table 3.2. Three observations that can be made here are that (1) SAMA is generally more compute/memory efficient than the baselines, (2) the cost of algorithmic adaptation for adaptive optimizers is marginal as expected, and (3) the efficiency gap further widens as we distribute compute/memory across multiple GPUs with our efficient DDP communication strategy. Since we were only able to conduct experiments with up to 4 V100 GPUs due to the limited compute resources, exploring extremely large-scale GBML with larger GPU servers remains a promising future research direction.

	GPUs	Memory	Throughput
Neumann	1	26.0	82.9
CG	1	28.4	82.1
SAMA-NA	1	13.7	144.1
SAMA	1	14.3	142.0
SAMA	2	10.4	241.2
SAMA	4	7.4	396.7

Table 3.2 Memory and throughput analysis on AGNews with 4 V100 GPUs.

To separately understand the effect of each component, we conduct a more extensive ablation by comparing test accuracy, GPU memory usage, and throughput of SAMA against various meta learning algorithms on IMDB and AGNews datasets from the Wrench benchmark. The results are provided in Table 3.3 & 3.4.

	Base Jacobian	**Algo Adapt**	**Distributed**	**Accuracy**	**Throughput**	**Memory**
Finetuning (no meta learning baseline)	x	x	x	85.79	169.16	7.77
Iterative Diff (*e.g.* MAML) [43, 105]	x	x	x	85.78	28.07	22.94
Conjugate gradient (*e.g.* iMAML) [130]	x	x	x	86.78	65.14	22.03
Neumann series [103]	x	x	x	86.65	67.03	19.70
DARTS (or $T_1 - T_2$) [99, 104]	o	x	x	86.36	43.69	10.81
SAMA-NA	o	x	x	86.55	137.90	10.30
SAMA	o	o	x	**89.05**	134.56	11.12
SAMA (2 GPUs)	o	o	o	**88.85**	226.27	8.00
SAMA (4 GPUs)	o	o	o	**89.02**	**298.28**	**6.46**

Table 3.3 Ablation results on AGNews

From our extended ablation study, it can be seen that 1) an identity approximation of base Jacobian significantly improves memory/compute efficiency, 2) algorithmic adaptation improves meta learning performance at the minimal compute/memory cost, and 3) our communication-optimized distributed training further improves compute/memory efficiency.

	Base Jacobian	Algo Adapt	Distributed	Accuracy	Throughput	Memory
Finetuning (no meta learning baseline)	x	x	x	78.16	144.39	6.60
Iterative Diff (*e.g.* MAML) [43, 105]	x	x	x	80.25	24.24	22.03
Conjugate gradient (*e.g.* iMAML) [130]	x	x	x	81.01	56.27	21.92
Neumann series [103]	x	x	x	79.92	57.85	19.75
DARTS (or $T_1 - T_2$) [99, 104]	o	x	x	80.47	37.53	10.35
SAMA-NA	o	x	x	81.92	117.86	9.93
SAMA	o	o	x	**84.31**	116.94	10.84
SAMA (2 GPUs)	o	o	o	**85.18**	196.48	7.84
SAMA (4 GPUs)	o	o	o	**84.19**	**263.74**	**6.39**

Table 3.4 Ablation results on IMDB

3.4.2 Continued Pretraining of Large Language Models

DAPT/TAPT [62] empirically demonstrate that additional pretraining (*i.e.*, continued pretraining) of the generic language model on the domain or task-specific data can further improve downstream performance on diverse benchmarks. However, the inclusion of low-quality samples for continued pertaining tasks can potentially hinder pretraining by amplifying negative interference [165], which could lead to suboptimal downstream performance. Here, we attempt to minimize such negative transfer by reweighting samples from the continued pretraining task with meta learning. To this end, we adopt the auxiliary learning technique from TARTAN [34] and simplify the two-stage pretraining-finetuning pipeline into a one-stage multitask learning pipeline with the reweighting scheme applied to the pretraining loss. The bilevel optimization formulation is as follows:

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, \mathscr{L}_{ft}(D_{ft}; \theta^*(\lambda))$$
$$\text{s.t. } \theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}}\, \mathscr{L}_{ft}(D_{ft}; \theta) + \frac{1}{|D_{pt}|} \sum_{x \in D_{pt}} w(x; \lambda) \cdot \mathscr{L}_{pt}(x; \theta)$$

where $\mathscr{L}_{ft}/\mathscr{L}_{ft}$ are finetuning/pretraining loss functions, D_{ft}/D_{pt} are finetuning/pretraining datasets, and $w(\cdot;\lambda)$ is the data reweighting network. Following the experiment setup in TARTAN [34], we use task-specific data and a masked language modeling loss in our auxiliary task and perform experiments with RoBERTa-base on 4 datasets from the original DAPT/TAPT paper. We compare our SAMA-based data optimization against DAPT and TARTAN-MT. We exclude TAPT and TARTAN-Meta respectively because (1) TAPT consistently underperforms TARTAN-MT [32] and (2) TARTAN-Meta uses additional validation data in the meta level of the downstream tasks, making the comparison unfair. We report our experiment results in Table 3.5.

As shown above, SAMA-based data optimization leads to improvements in downstream performance on almost all datasets. This indirectly demonstrates that SAMA-based data reweighting can identify more/less relevant data in the auxiliary task and accordingly up-/down-weight them, unlike TARTAN-MT which allocates equal importance weights on all auxiliary data. Therefore, we expect that our method would likely benefit from additional auxiliary data by automatically figuring out and exploiting only relevant data, whereas TARTAN-MT is much more susceptible to negative transfer. While we only used task-

	ChemProt	HyperPartisan	ACL-ARC	SciERC	Average
Baseline	82.70 (0.45)	89.03 (2.25)	68.17 (2.52)	79.83 (0.89)	79.93
DAPT [62]	84.17 (0.50)	87.23 (3.65)	71.84 (4.78)	80.42 (1.57)	80.92
TARTAN-MT [34]	84.18 (0.30)	94.64 (0.91)	**72.41** (1.94)	80.83 (0.71)	83.02
SAMA (ours)	**84.49** (0.13)	**95.18** (0.03)	71.63 (1.68)	**81.84** (0.08)	**83.29**

Table 3.5 Experiment results for auxiliary learning with the continued pretraining task. Following [62], we report test micro-F1 for ChemProt and macro-F1 for the other datasets. The number in parentheses indicates the standard deviation for each experiment over 3 runs.

specific data in our auxiliary task for the fair comparison with TARTAN-MT, extending auxiliary data to domain-specific or even general text data and comparing SAMA against DAPT or TARTAN-MT would be an intriguing future research direction. Finally, we analyze the GPU memory usage of different-sized RoBERTa in this experiment and present the result in Figure 3.1. The figure clearly shows the superior memory efficiency of SAMA with the increasing model size.

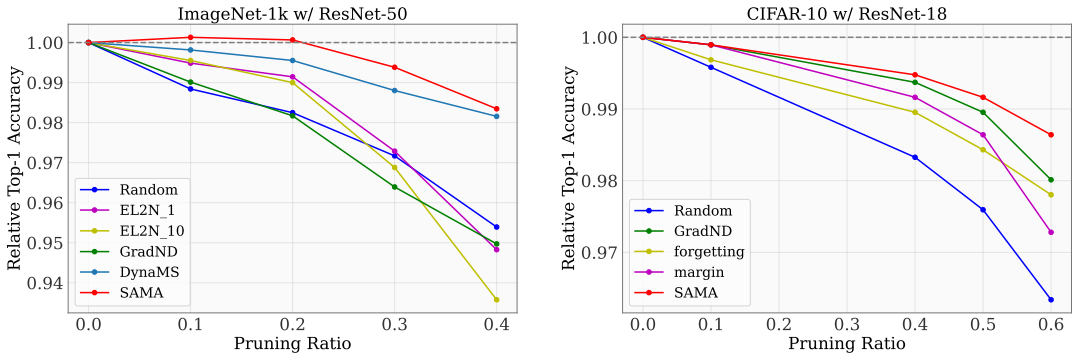
3.4.3 Scale-Agnostic Efficient Data Pruning

Data pruning [119, 150, 155, 162] has recently received the limelight in the machine learning community as a means to both improve training efficiency and reduce (semantic) redundancy in training data. In particular, Sorscher et al. [150] showed both theoretically and experimentally that neural scaling laws can be beaten by data pruning. Nevertheless, they point out that the optimal data pruning metric varies across different dataset scales and further research in scalable data pruning metrics is needed. Here, we propose to forgo hand-designed data pruning metrics, and rather automatically *meta-learn* the importance weight of each training data following Meta-Weight-Net (MWN) [145] with four major modifications. First, we replace their iterative differentiation meta gradient algorithm with SAMA to achieve improved memory/compute efficiencies. Second, we further speed up meta learning by enabling distributed training with our efficient communication strategy. Third, we use the uncertainty of the prediction in addition to the loss value as an input to MWN to better estimate importance weight of each training data. Last, we use training data *both* in the base and the meta levels, assuming no additional validation data. A bilevel optimization formulation of our method is as follows:

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\ \mathscr{L}(D_{train}; \theta^*(\lambda))$$
$$\text{s.t. } \theta^*(\lambda) = \underset{\theta}{\operatorname{argmin}}\ \frac{1}{|D_{train}|} \sum_{(x,y)\in D_{train}} w(\mathscr{L}, \mathscr{U}; \lambda) \cdot \mathscr{L}(x, y; \theta)$$

where $w(\cdot;\lambda)$ is MWN that takes the loss value $\mathscr{L}$ and the uncertainty $\mathscr{U}$ of the training sample (x,y) as an input and outputs the importance weight. Under this setup, we run meta learning with SAMA for 30 / 50 epochs respectively for ImageNet-1k / CIFAR-10 and obtain the pruning metrics by averaging the importance weights of the last 5 epochs. We compare

our method to several popular static/dynamic data pruning baselines, and present the results in Figure 3.4.



	$EL2N_1$ [119]	$EL2N_{10}$ [119]	$GradNd_{10}$ [119]	DynaMS [162]	SAMA (ours)
Relative Search Time	0.16	1.65	1.65	0.16	0.46

Fig. 3.4 **Top Left:** ImageNet-1k data pruning results with ResNet-50. Reported numbers are relative accuracy compared to full training accuracy (*i.e.*, pruned_acc/full_acc). Accuracy for other baseline methods is obtained from DynaMS [162]. **Top Right:** CIFAR-10 data pruning results with ResNet-18. Accuracy for other baseline methods is obtained from Deepcore [59]. **Bottom:** Relative time spent in finding data to prune compared to full ImageNet-1k training time.

As expected, GBML-based data pruning with SAMA not only outperforms heuristics-based data pruning but also works well across different dataset scales. Surprisingly, we observe that GBML-based data pruning even leads to improvements in test accuracy at the pruning ratio of 0.1 and 0.2 on ImageNet-1k. The potential implication is that ImageNet-1k may have noisy labels or semantic redundancy and that GBML is able to automatically figure and filter out these samples. Further in-depth investigation of filtered data remains an interesting research direction. Considering that compute/memory inefficiency has traditionally been the major bottleneck in GBML applications, we also compare the relative search time for data pruning. Our result shows that SAMA demonstrates comparable or even shorter search time than heuristics-based methods. We also note that, while the original MWN [145] encounters the OOM error under our setup of batch_size=256, the throughput analysis with the reduced batch size reveals that efficient distributed training with SAMA on 4 GPUs achieves 15-20$\times$ speed up compared to the original MWN that lacks distributed training support.

3.4.4 The Effect of Scaling in Model-Agnostic Meta Learning

Since the inception of MAML [43], a myriad of algorithms have been proposed to improve few-shot image classification while assuming a fixed network architecture. In contrast, here we shift our focus from the algorithm to the scale, and propose to study the following question: "Leveraging the compute/memory efficiency of SAMA, can we improve the few-shot generalization capability by scaling up the network size?". Since SAMA is a

variant of implicit differentiation, we closely follow the experiment setup in iMAML [130], where proximity to the initialization weights is explicitly enforced by L_2-regularization. The major difference is that iMAML uses a conjugate-gradient-based method, which requires second-order gradient information to compute meta gradients, while we adopt SAMA to achieve improved scaling to larger networks with its superior memory/compute efficiency. We conduct preliminary experiments on the Omniglot 20-way 1-/5-shot tasks with the basic 4-layer CNN architecture, while varying the width (hidden size) of the networks to study the effect of the model size on the few-shot classification accuracy. The experiment results are provided in Figure 3.5 below.

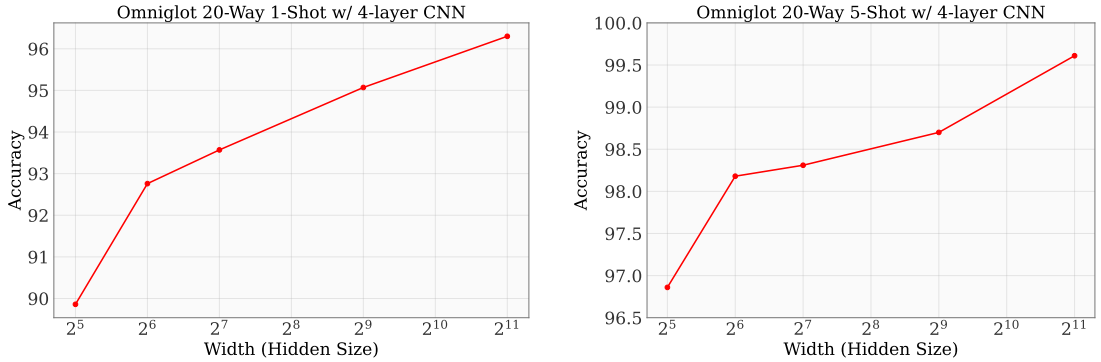


Fig. 3.5 Few-shot image classification accuracy on Omniglot 20-way 1-/5-shot tasks with varying network sizes.

Interestingly, we observe that the increased model size leads to consistent improvements in few-shot classification accuracy. The important question following this observation is "can we apply scaling laws [84] from other tasks (*e.g.*, language modeling) to general meta learning beyond few-shot image classification?" Since meta learning involves two optimization problems (meta and base) unlike traditional machine learning problems, it is as of now unclear how to define the general concept of "scale" in terms of both model and dataset sizes. We expect that further research in this direction would be critical in systematically studying scalable meta learning.

3.5 Related Work

Algorithms Two major lines of research in gradient-based meta learning algorithms are iterative and implicit differentiation [53]. Iterative differentiation [43, 45, 46, 105] computes meta gradients by differentiating through the optimization path and therefore requires saving all intermediate states on the path. This makes the memory/compute costs of iterative differentiation increase linearly with the number of unrolling steps. While the linear cost can be avoided with the use of techniques like truncated backpropagation [95], it is still more expensive than that of implicit differentiation, in which meta gradient computation is independent of the length of the optimization path. More specifically, meta gradient computation in implicit differentiation depends *only* on the final state of the optimization

path. To compute base Jacobian inversion in implicit differentiation, a multitude of variants have been proposed, each of which uses Neumann series [25, 103], conjugate gradient [130], Nystrom method [66], and more. While generally being more compute/memory efficient than iterative differentiation, most existing implicit differentiation algorithms have poor scalability due to the issues studied in Sec. 3.3.

Applications Meta learning has found many applications in machine learning including few-shot learning [43, 130, 182], neural architecture search [99, 178], hyperparameter optimization [45, 46, 103, 105], data optimization [58, 136, 145, 180], and reinforcement learning [61, 76, 132], to name a few. Notably, most of these applications share the underlying mathematical formulation of bilevel optimization and are conceptually related to optimal design and inductive/declarative programming paradigms.

Systems Compared to algorithms and systems research, there are relatively fewer research efforts in meta learning systems. In an attempt to facilitate research in few-shot image classification, *higher* [54], *learn2learn* [5], and *TorchMeta* [32] have been developed. However, due to their specific focus on few-shot image classification, these libraries have not been actively used in other meta learning tasks, such as data optimization or neural architecture search. Recently, software libraries for implicit differentiation including *JaxOpt* [18] and *Betty* [24], have been proposed. Given that Betty's software architecture is specifically designed to support various systems optimization for large-scale meta learning, we chose to implement SAMA in this framework.

3.6 Conclusion

In this paper, we strived to make scalable meta learning practical via both algorithmic and systems advancements. Towards this goal, we investigated diverse scaling bottlenecks in meta learning at a technical level and resolved them by developing SAMA. Tested on multiple benchmarks, SAMA empirically demonstrated its scaling efficiency as well as its capability to optimize a variety of high-dimensional inductive biases of large-scale learning. In future work, we plan to explore two directions. First, given that training extremely large models with 10B+ parameters require various systems techniques such as model/pipeline parallelism or optimizer sharding, extending SAMA to be compatible with these techniques would be highly important for further scalability. Second, we will also focus on large-scale meta learning application research, such as neural architecture search for Transformer-family models. Overall, we hope that our work can serve as a stepping stone for a lot of interesting scalable meta learning research to come.

Interpretability: Data Influence Analysis

Chapter 4

ML Debugging Systems

In Section 1.1.3, we showed that the investigation of variants of influence functions enable a multitude of analyses of ML models, including training data attribution and uncertainty estimation. Nevertheless, applying influence functions to recent LLMs and their vast training datasets has been largely limited by prohibitive compute and memory costs. To address this, this section aims to significantly improve scalability of influence functions with an efficient gradient projection strategy called LOGRA that leverages the inherent gradient structure in backpropagation. We then provide a theoretical motivation of gradient projection approaches to influence functions to promote trust in the data influence analysis. Lastly, we lower the barrier to implementing data influence analyses by introducing LOGIX, a software package that can transform existing training code into data valuation code with minimal effort.

4.1 Introduction

Despite the well-recognized importance of training data in advancing the capabilities of large language models (LLMs) [19, 84, 135], there is no agreed-upon mechanisms for crediting or compensating data providers. As LLMs are increasingly integrated into our society and economy, the absence of such mechanisms has aggravated a tension between data and model providers, exemplified by recent legal challenges involving major tech companies [82, 108]. In this atmosphere, data valuation, which quantifies the contribution of each training data to the model output, has been discussed as a potential technical solution for tackling these societal issues [42, 51, 74, 81, 167, 179].

At a high level, most data valuation algorithms interpret the model output as a coalition of its training data, and evaluate the contribution of each example based on its influence on the model output when included or excluded from the training dataset [51, 77, 87, 92]. If an inclusion of a specific training example consistently improves model performance, high value can be assigned to this example for its contribution. However, applying existing data valuation methods to recent LLMs and their vast training datasets has faced significant scalability challenges to date. For instance, sampling-based methods, such as the Shapley value [51, 92] or Datamodels [77], require retraining the model multiple times with varied combinations of data subsets to directly model the effect of in/excluding each data. Unfortunately, such

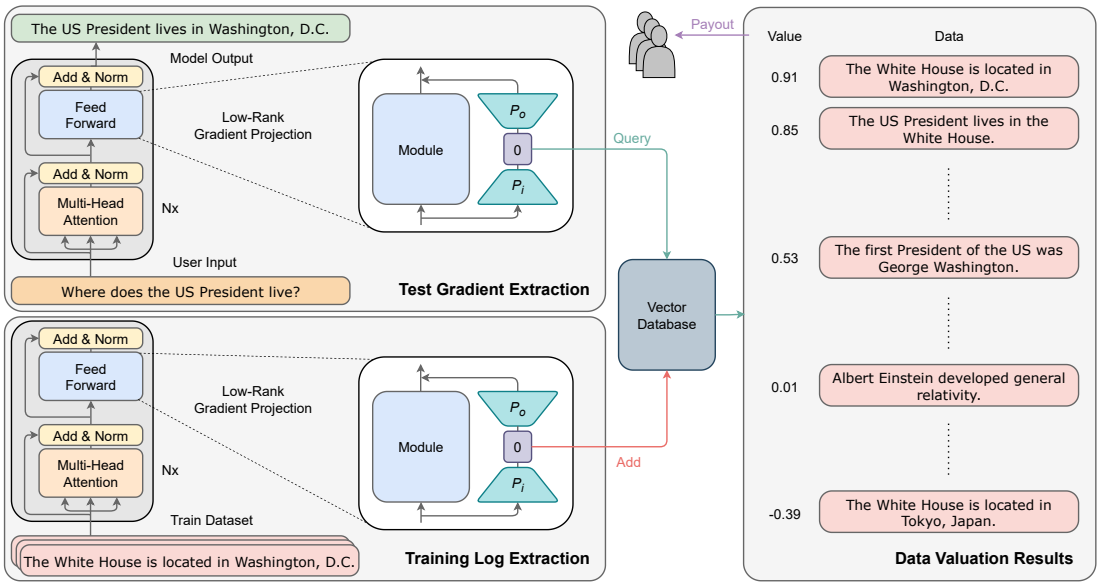


Fig. 4.1 Data valuation system architecture. **(Left Bottom)** We first extract the Hessian and gradients for all training data using efficient gradient projection LOGRA and store them in a database. **(Left Top)** At test time, we similarly extract gradients and query the database. **(Right)** The database returns similarity scores with respect to training examples that can be used for data valuation/attribution.

repeated retraining is hardly affordable even for small models, let alone LLMs. To overcome this issue, gradient-based methods, including influence functions [87, 117], approximate the effect of data in/exclusion on the model output using gradient information without costly retraining. Even so, scaling gradient-based methods to LLMs is hindered by prohibitive compute and memory costs originating in the high-dimensional nature of the gradient.

Consequently, the main objective of this work is to bridge the gap in scaling existing data valuation methods to recent LLMs and their vast training datasets. Toward this goal, we focus on influence functions [87, 117], a representative gradient-based data valuation method, and significantly improve its scalability with an efficient gradient projection algorithm. We visualize the proposed data valuation system in Figure 4.1, and detail our technical contributions below:

- Employing gradient structures in backpropagation, we develop a novel **lo**w-rank **gra**dient projection algorithm LOGRA that improves space & time complexity of gradient projection, a major scalability bottleneck in prior work [117, 142], from $O(nk)$ to $O(\sqrt{nk})$ where n and k are model and projection dimensions. Furthermore, LOGRA directly computes projected gradients without materializing full gradients, enabling low GPU memory and high GPU utilization for improved efficiency. Lastly, we show that LOGRA can be easily implemented with small add-on layers, similarly to LoRA [73].

- By interpreting a damping term in influence functions as a spectral gradient sparsification mechanism, we (1) offer a theoretical motivation of gradient projection approaches to influence functions and (2) derive a specialized PCA initialization scheme for LOGRA.

- We introduce software named LOGIX that (1) makes it *simple* to convert existing training code into data valuation code, (2) is *compatible* with various scalability tools and features in the LLM ecosystem, and (3) is *extensible* to implement other data valuation or interpretability algorithms.
- In our data valuation experiments, LOGRA demonstrates competitive accuracy against more costly baselines, while showing up to 6,500× increase in throughput and 5× reduction in GPU memory, when applied to Llama3-8B-Instruct [2] and the 1B-token dataset, compared to EKFAC influence [55], the state-of-the-art and only runnable baseline at this scale. We also observe that most valuable data identified by LOGRA generally share qualitative similarities with the queried LLM output.

4.2 Scalability Bottlenecks in Influence Functions

Most data valuation algorithms (*e.g.*, data Shapley [51]) evaluate the contribution or value of a specific example x on the utility v (*e.g.*, test loss), that can further be used for crediting data providers, by measuring the overall change in the utility v when in/excluding x as follows:

$$\textsc{Value}(x;v) = \sum_{S \subseteq D \setminus \{x\}} w\big(v(S \cup \{x\}) - v(S)\big) \tag{4.1}$$

where D is the training dataset, S is a subset of D, and w is an (algorithm-specific) weighting term. Intuitively, the larger the utility gain from an inclusion of x is, the larger the value of x is.

One popular instantiation of Eq. (4.1) is the leave-one-out error [87], a semivalue [38] that is a basis for most data attribution methods and only considers S with $|S| = |D| - 1$ (*i.e.*, leaving one example x from the entire dataset D). However, naively computing the leave-one-out-error requires retraining the model multiple times for each $x \in D$, which is hardly affordable even in small-scale setups. To overcome this issue, influence functions, a representative gradient-based method, efficiently *simulates* the effect of model retraining without an example x_{tr} on the utility using gradient information as:

$$\textsc{Influence}(x_{tr}, x_{te}) = g_{te}^{\top} H^{-1} g_{tr} \tag{4.2}$$

where g_{tr} and g_{te} are train and test gradients respectively, and H is the Hessian matrix. Concretely, influence functions approximate the effect of removing x_{tr} by updating the model parameters with a Newton step in the direction of $H^{-1}g_{tr}$, and uses a first-order Taylor approximation to estimate how this update will affect the test utility. In practice, computing influence functions involves two key steps of (1) solving the inverse Hessian-vector product (iHVP) with g_{te}, and (2) taking the dot product of this iHVP with the gradient g_{tr} for each training example.

Despite their comparative efficiency, influence functions remain difficult to scale to recent LLMs due to the high compute and memory costs associated with both steps. First, space

and time complexity of naive iHVP are respectively $O(n^2)$ and $O(n^3)$, both of which are impractical in recent LLMs with $n > 10^9$ parameters. To address this issue, various tricks for efficiency, such as iterative methods [87] or EKFAC approximation [55], have been proposed. Second, to ensure fair valuation, one must compute influence scores with *all* training data, which requires access to their gradients. However, computing gradients for all training data approximately amounts to one-epoch training, the cost of which often exceeds $1M in the context of LLM (pre)training. If training gradients were to be recomputed frequently for regular data valuation, the total cost can quickly become astronomical. Thus, while it is technically possible to run a few influence function analyses to interpret interesting LLM outputs using efficient iHVP tricks [55], doing it in a scalable and sustainable way to build a practical data valuation system remains a significant challenge.

In an attempt to mitigate the aforementioned cost issues, Arnoldi IF [142] and TRAK [117] recently explored the strategy of projecting gradients onto a low-dimensional space and computing influence scores on the subspace spanned by the projection matrix as follows:

$$\textsc{Influence}(x_{tr}, x_{te}; P) = \left(Pg_{te}\right)^{\top} \left(PHP^{\top}\right)^{-1} \left(Pg_{tr}\right) \tag{4.3}$$

where $P \in \mathbb{R}^{k \times n}$ is the projection matrix given the model and projection dimensions of n and k. Under this strategy, the iHVP operation also occurs in a low-dimensional space, meaning that n in memory and compute complexity of iHVP gets replaced with $k \ll n$. Furthermore, low-rank projection enables writing projected gradients for all training data to disk once and simply reading them as new test data arrives without costly re-computations. This converts an influence function problem into a vector similarity search problem, for which various system optimizations exist [83].

In essence, this strategy significantly reduces both iHVP and training gradient recomputation costs by introducing an additional process of low-rank gradient projection Pg. However, the additional compute/memory costs and accuracy degradation incurred from low-rank gradient projection has not been thoroughly studied to date. First, assuming that the batch size is b, the compute cost of naive batched gradient projection is $O(bkn)$. Noting that the compute cost of backpropgation is $O(bn)$ (or $O(btn)$ if we consider the time dimension), the cost of gradient projection is usually larger than that of backpropagation given a reasonably large k for the expressivity. Second, the memory costs for full per-sample gradient and the projection matrix are $O(bn)$ and $O(kn)$. If an 8B model were to be used, each of these costs amounts to 32GB$\times b$ (or $\times k$) GPU memory. While Arnoldi IF and TRAK attempt to address the memory costs of the per-sample gradient and projection matrix respectively with forward-mode Jacobian-vector products and a custom CUDA kernel trick, neither of them are able to solve both issues altogether. This leads Arnoldi IF and TRAK to use very small k and b, each of which results in decreased accuracy of influence scores due to limited expressivity and poor efficiency from low GPU utilization. Since accuracy and efficiency are both critical for effective data valuation, we deduce that further advancements in the gradient projection approach are necessary.

4.3 Scaling Data Valuation & Influence Functions

In light of these issues, we first design a memory and compute efficient gradient projection algorithm called LoGra, that leverages the inherent gradient structure in backpropagation (Section 4.3.1). Then, we provide an intuitive theoretical analysis on why gradient projection approaches work in influence functions (Section 4.3.2). Finally, we distill our insights obtained from studying (scalable) influence functions into a new open-source software, called LogIX, which achieves high compatibility, extensibility, and usability, to facilitate data valuation research (Section 4.3.3). In this section, we build our arguments at the granularity of each layer (or module) instead of the whole network for clarity.

4.3.1 Algorithm: Memory and Compute Efficient Gradient Projection

Most layers in neural networks, such as linear and convolutional layers, essentially perform matrix multiplication. Given the input $x_i \in \mathbb{R}^{n_i \times T}$, the output $x_o \in \mathbb{R}^{n_o \times T}$, the weight $W \in \mathbb{R}^{n_o \times n_i}$ for the layer, its forward and backward computations can be written as follows:

$$\textbf{Forward:} \qquad x_o = W x_i \tag{4.4}$$

$$\textbf{Backward:} \qquad \mathrm{vec}(\mathscr{D}W) = \sum_{t=1}^{T} x_{i,t} \otimes \mathscr{D}x_{o,t}\,, \quad \mathscr{D}x_i = W^\top \mathscr{D}x_o \tag{4.5}$$

where T denotes for the sequence dimension in language modeling, $\mathscr{D}$ the derivative with respect to the loss, $\otimes$ the Kronecker product, and $\mathrm{vec}(\cdot)$ the vectorization operation. In Eq. (4.5), we observe that gradient $\mathrm{vec}(\mathscr{D}W)$ obtained during backpropagation is structured as a sum of Kronecker products between forward and backward activations. LoGra leverages this observation to impose an additional Kronecker-product structure on the projection matrix P as follows:

$$P\mathrm{vec}(\mathscr{D}W) \triangleq (P_i \otimes P_o)\mathrm{vec}(\mathscr{D}W) = \sum_{t=1}^{T} (P_i \otimes P_o)(x_{i,t} \otimes \mathscr{D}x_{o,t}) = \sum_{t=1}^{T} P_i x_{i,t} \otimes P_o \mathscr{D}x_{o,t} \tag{4.6}$$

where $P_i \in \mathbb{R}^{k_i \times n_i}$, $P_o \in \mathbb{R}^{k_o \times n_o}$, and $P = P_i \otimes P_o$. In Eq. (4.6), LoGra first projects forward and backward activations onto low-dimensional spaces with P_i and P_o respectively, and then reconstructs projected gradient directly from these projected activations. This is in contrast to traditional gradient projection [117], which first computes raw gradient and then projects it onto a low-dimensional space.

Now, we compare memory/compute efficiency of LoGra to that of naive gradient projection, especially under the setting of $n_i \approx n_o \approx \sqrt{n}$ and $k_i \approx k_o \approx \sqrt{k}$. First, both memory/compute costs of per-sample gradient computations reduce from $O(bn)$ to $O(bk)$. Second, both memory/compute costs of gradient projection reduce from $O(bnk)$ to $O(b\sqrt{nk})$. To clearly see this benefit, given the model/projection sizes of 8B/4k, we note that projection matrix sizes are about 1GB and 128TB respectively for LoGra and naive projection. As such, while enjoying general efficiency gains from gradient projection we disscussed in

Section 4.2, LOGRA further improves the efficiency of per-sample gradient computations significantly at a marginal cost of the additional gradient projection process.

Furthermore, leveraging the fact that projection occurs in the activation space, LOGRA can be easily implemented with small add-on layers that are composed of *encoder*, *bottleneck*, and *decoder*, each of which is initialized with P_i, zero, and P_o as shown in Figure 4.2. If we ignore the bottleneck layer, the overall architecture is identical to the popular LoRA architecture [73]. While it is intuitive that the roles of encoder and decoder are projecting forward and backward activations respectively, we emphasize two critical roles of the bottleneck layer here. First, its zero initialization ensures that the rest of both forward and backward computations remain unaffected by these add-on layers. Second, per-sample projected gradients can be obtained by simply computing per-sample gradients for the bottleneck layer, using automatic differentiation of an underlying framework without complicated implementation efforts.

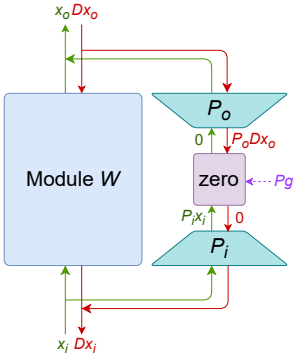


Fig. 4.2 LOGRA.

4.3.2 Theory: Why Gradient Projection Works in Influence Functions

While LOGRA can significantly improve scalability of influence functions, an inherent criticism of any gradient projection approach is that information loss from the projection process may render the resulting influence analysis invalid. Unfortunately, theoretical analyses from prior work [117, 142] only discuss the indirect effect of gradient projection on proxy concepts like gradient flow or iHVP variance, which are loosely related to influence functions. To promote trust in the data valuation process, we provide here a mathematical motivation of gradient projection approaches to influence functions. Toward this goal, we interpret a damping term in influence functions that is typically added to ensure the invertibility of the Hessian H as a *spectral gradient sparsification* mechanism. To formalize the above claim, we here first provide two required assumptions, and then theoretically show the spectral gradient sparsification effect of a damping term in influence functions below.

Assumption 1 *In this work, we make the following two assumptions on train & test gradient distributions and the Hessian H:*
1. Given that language modeling falls under the maximum likelihood framework, we replace the Hessian H with the Fisher Information Matrix (FIM), and further approximate the FIM with the empirical FIM, i.e.,

$$\begin{aligned} H &= \mathbb{E}_{p_\theta(y|x)}\left[\nabla \log p_\theta(y|x) \nabla \log p_\theta(y|x)^\top\right] \\ &\approx \frac{1}{N} \sum_{(x_n,y_n)\in D_{tr}} \left[\nabla \log p_\theta(y_n|x_n) \nabla \log p_\theta(y_n|x_n)^\top\right] \end{aligned}$$

2. Given that test data are directly sampled from the model given the prompts, we assume test gradients g_{te} and train gradients g_{tr} approximately follow the same distribution.

Lemma 1 *Let* $\{e_1, \cdots, e_n\}$ *and* $\{\lambda_1, \cdots, \lambda_n\}$ *be eigenvectors and eigenvalues of the Hessian* H*. Expressing* $g_{tr/te} = \sum_i c_{tr/te,i} \cdot (\sqrt{\lambda_i} e_i)$*, the following holds under Assumption 1:*

$$Influence(x_{tr}, x_{te}) = g_{te}^\top (H + \lambda I)^{-1} g_{tr} = \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + \lambda} c_{tr,i} c_{te,i} \;\; and \;\; \mathbb{E}[c_{\cdot,i}^2] \approx 1.$$

Proof.
Let $Q = [e_1, \cdots, e_n]$ and $\Lambda = diag(\lambda_1, \cdots, \lambda_n)$.

$$\begin{aligned}
\mathrm{IF}(x_{tr}, x_{te}) &= g_{te}^\top (H + \lambda I)^{-1} g_{tr} \\
&= g_{te}^\top (Q \Lambda Q^\top + \lambda I)^{-1} g_{tr} \\
&= g_{te}^\top \left(Q(\Lambda + \lambda I) Q^\top\right)^{-1} g_{tr} \\
&= g_{te}^\top Q (\Lambda + \lambda I)^{-1} Q^\top g_{tr} \\
&= \left(\sum_i c_{te,i} \cdot (\sqrt{\lambda_i} e_i) \right)^\top Q (\Lambda + \lambda I)^{-1} Q^\top \left(\sum_i c_{tr,i} \cdot (\sqrt{\lambda_i} e_i) \right) \\
&= \left[c_{te,1} \sqrt{\lambda_1}; \cdots; c_{te,n} \sqrt{\lambda_n} \right]^\top (\Lambda + \lambda I)^{-1} \left[c_{tr,1} \sqrt{\lambda_1}; \cdots; c_{tr,n} \sqrt{\lambda_n} \right] \\
&= \sum_{i=1}^{n} \frac{\lambda_i}{\lambda_i + \lambda} c_{tr,i} c_{te,i}
\end{aligned}$$

□

Since we assume g_{te} and g_{tr} follow the same distribution, we need to show $\mathbb{E}[c_{tr,i}^2] \approx 1$ for all i.

$$\begin{aligned}
\Lambda &= Q^\top Q \Lambda Q^\top Q \\
&= Q^\top H Q \\
&\approx \frac{1}{N} \sum_{(x_i, y_i) \in D_{tr}} Q^\top \left[\nabla \log p_\theta(y_n | x_n) \nabla \log p_\theta(y_n | x_n)^\top \right] Q \quad \text{(Assumption 1)} \\
&= \mathbb{E}\left[Q^\top g_{tr} g_{tr}^\top Q \right] \\
&= \mathbb{E}\left[Q^\top \left(\sum_i c_{tr,i} \cdot (\sqrt{\lambda_i} e_i) \right) \left(\sum_i c_{tr,i} \cdot (\sqrt{\lambda_i} e_i) \right)^\top Q \right] \\
&= \mathbb{E}\left[\left[c_{tr,1} \sqrt{\lambda_1}; \cdots; c_{tr,n} \sqrt{\lambda_n} \right] \left[c_{tr,1} \sqrt{\lambda_1}; \cdots; c_{tr,n} \sqrt{\lambda_n} \right]^\top \right]
\end{aligned}$$

Inspecting diagonal terms, we get $\lambda_i \approx \mathbb{E}[c_{tr,i}^2 \lambda_i] = \mathbb{E}[c_{tr,i}^2] \lambda_i$.

Therefore, $\mathbb{E}[c_{tr,i}^2] \approx 1$. □

Lemma 1 shows that a damping term *softly* limits the number of components in influence computations by penalizing contributions from small components. Given the prevalence and practical importance of a damping term in influence functions [13], we can motivate gradient

projection as an alternative way of (hard-)limiting influence computations to components in the projection matrix. To make LOGRA similarly penalize small components, we develop an initialization scheme that exploits the Kronecker-Factored Approximate Curvature (KFAC) algorithm [106]. As a quick overview, KFAC approximates the block-wise Hessian with the Kronecker product of uncentered forward and backward covariances of each layer, respectively denoted with C_F and C_B, as $H \approx H_{KFAC} = C_F \otimes C_B$. Expressing C_F and C_B as $Q_F \Lambda_F Q_F^\top$ and $Q_B \Lambda_B Q_B^\top$ with eigendecomposition, it is easy to show that eigenvectors and eigenvalues of H_{KFAC} are $Q_F \otimes Q_B$ and $\Lambda_F \otimes \Lambda_B$. Consequently, we can approximately discard the smaller components of H by initializing P_i and P_o with $Q_F^{1:k_i}$ and $Q_B^{1:k_o}$, where $Q^{1:k}$ is a collection of top-k eigenvectors (similar to performing PCA on forward and backward activations). In Section 4.4, we experiment with both PCA and random initialization schemes.

4.3.3 Software: Compatibility, Extensibility, and Usability

Besides algorithmic efficiency, another major bottleneck in the practical adoption of data valuation systems is often the challenge of implementation. In particular, we observe that gradient computation in LLMs, which is a building block for influence functions, typically requires support from other scalability tools like DeepSpeed [133] or relies on high-level frameworks like HF Transformers [166]. However, most existing software that can be used for data valuation (*e.g.*, Captum [88] and TRAK [117]) is largely incompatible with these tools due to the (too) high level of abstraction in their APIs.

```
import logix

# setup
run = logix.init(project, config)
run.setup("stat": "kfac", "save": "grad")
run.watch(model)

# train log & statistic
for batch in train_loader:
  with run(data_id=batch["input_ids"]):
    loss = model(batch)
    loss.backward()
run.finalize()

# test time influence analysis
with run(data_id=tst_batch["input_ids"]):
  loss = model(tst_batch)
  loss.backward()
run.compute_influence_all()
```

Listing 4.1 Code Example of LOGIX.

Subsequently, we develop a new software package, LOGIX, design of which enables an *easy conversion* of users' existing training code into data valuation code, by promoting compatibility with other tools in the LLM ecosystem. To this end, we first notice that most influence function algorithms simply require collecting train logs (*e.g.*, gradient, activation) and their statistics (*e.g.*, covariance). As a result, given arbitrary users' training code,

data valuation software only need to intercept these logs, and provide basic primitives to compute various statistics with them. Leveraging this observation, LOGIX implements log interceptions and compute primitives using PyTorch hooks. Notably, the use of hooks makes LOGIX *compatible* with diverse other tools as hooks can be seamlessly integrated with most PyTorch features (*e.g.*, FSDP, autocast, compile). In addition, LOGIX is *extensible*, as users can easily define and add custom primitives inside hooks. Finally, LOGIX is *easy-to-use* as its context manager automatically handles adding appropriate hooks and primitives to relevant modules with minimal code changes. Code examples can be found in Figure 4.1 and our project page.

Difference between LOGIX and Other Interpretability Tools

Influence functions have been extensively studied as an interpretable AI method. Accordingly, there have been several tools originating in the AI interpretability field that implement influence functions, with most notable examples including Captum [88], TRAK [117], and Kronfluence [55]. Overall, the software design of these tools aim at easing the *from-scratch implementation* of influence functions by introducing a lot of abstraction, following the philosophy of high-level frameworks. In fact, such software designs were well-received in the pre-LLM era. Nonetheless, as scaling has become a key aspect of AI research, the (LLM) development ecosystem has become complicated and being able to compatibly work with other tools in the ecosystem has become a core aspect in the ML software design. Hence, unlike existing software, the design of LOGIX aims at enabling the *easy conversion* of users' (already efficient) training codes into data valuation codes. This design is also motivated by the observation that gradient is simply a by-product of the training procedure so that we can reuse most of the training code for data valuation without needing to write the gradient computation code from scratch as in other tools.

Recently, there have been active developments in (mechanistic) interpretability software, represented by TransformerLens [113] and pyvene [168]. Interestingly, these software also extensively use PyTorch hooks, similarly to LOGIX, probably due to its high compatibility with other features such as autocast, distributed data parallelism, fully-sharded data parallelism, and gradient checkpointing. Nevertheless, we point out two major differences between these (mechanistic) interpretability software and LOGIX. First, support for dataset-level statistics computations in LOGIX is largely missing in these tools. In data valuation, we often need to compute several dataset-level statistics such as the Hessian (or Fisher information matrix) for accurate influence computations, and therby supporting these computations seamlessly was an important design principle behind LOGIX. However, analyses in (mechanistic) interpretability research typically focuses on each instance and computing dataset-level statistics is typically not supported. Second, support for efficient data IO in LOGIX is not a priority in other tools. As we propose to convert the data valuation problem into a vector similarity search problem with gradient projection, we put efforts into improving efficiency of data IO (see the next subsection for details), whereas this issue is rarely considered in other interpretability tools. We hope to explore the possibility of supporting both data valuation and other interpretability research in a unified way with LOGIX as our future work.

Optimizations in LOGIX

Efficient Data IO With LOGRA, we propose to save projected gradients for *all* training data to disk, and frequently load them as a new test batch arrives. As a result, reducing latency from data IO renders to be critical in realizing efficient data valuation. In particular, as the total size of all training gradients is usually far beyond the limit of CPU memory, we should optimize data transfer between disk and CPU (or GPU). To address this issue, we adopted the memory-mapped files that bypasses the need for intermediate copying between kernel space and user space, reducing the overhead associated with data IO operations. The use of the memory-mapped files is also motivated by the observation that, given each query batch, data valuation often requires computing influence scores with all training data. Therefore, we can access training gradients in a predefined or sequential order instead of in a random order, which can be done efficiently with memory-mapped files (sequential access is faster than random access).

Moreover, we overlap memory-mapped-file-based data IO with computations to further enhance data valuation efficiency. In the logging phase, we overlap the process of saving gradients extracted from the current training batch to disk with computations for the next training batch using Python multiprocessing. In the influence computation phase, we overlap the process of loading saved training gradients from disk with computing a dot product with the query batch using the pre-fetching feature of PyTorch DataLoader.

We also note that more efficient data IO can be achieved by the use of more advanced techniques like GPU-accelerated vector database, especially in the production setting. While we considered supporting this feature, we decided to focus on the memory-mapped-file-based data IO in our initial version of LOGIX, as it offers more flexibility to explore different algorithms in the research setting.

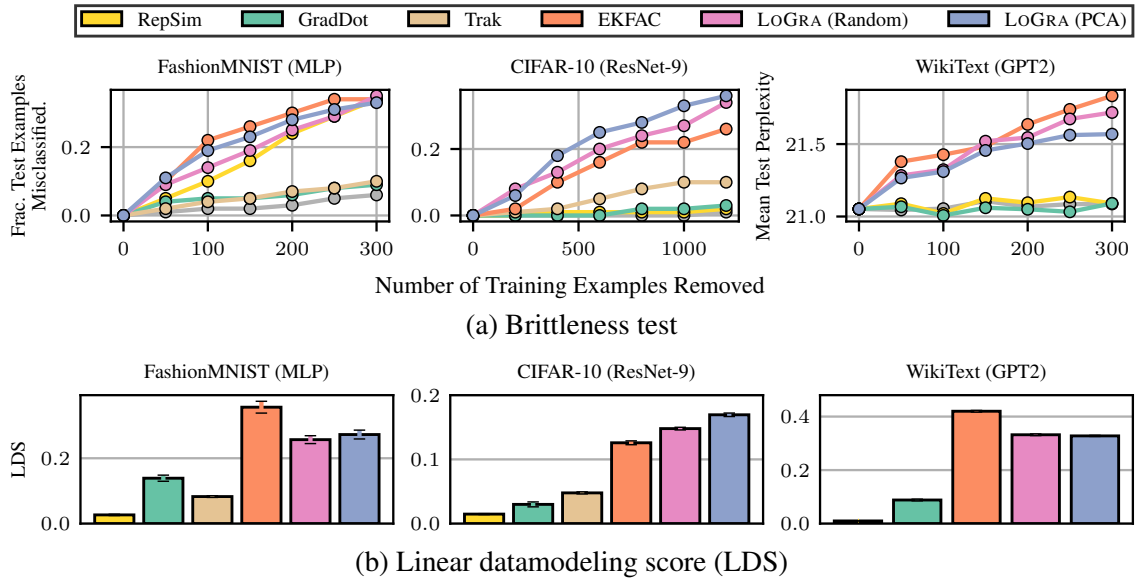
Memory Optimization When dealing with LLMs, GPU memory is often a major scaling bottleneck. To alleviate this issue, we support CPU offloading of dataset-level statistics by utilizing the sequential nature of backpropgation. When this feature is enabled, we by default keep all dataset-level statistics (*e.g.*, gradient covariance) on CPU, move it to GPU when the corresponding module is called during forward/backward passes, and then move it back to CPU asynchronously as soon as updating statistics for the module is done. Depending on the CPU-GPU communication bandwidth, this feature may slow down the logging process.

Communication Optimization If training data are split across multiple processes with distributed training, we need to aggregate dataset-level statistics across processes for consistency. To minimize the communication cost, we delay the synchronization process until the training loop (one epoch) is over, and perform synchronization only once at the end. Following the similar logic, users can maximize the efficiency of the logging phase by disabling gradient synchronization (*e.g.*, torch.no_sync).

4.4 Experiments

In this section, we evaluate the effectiveness of LoGra in terms of *accuracy* and *efficiency*, both of which are important in practical data valuation systems. Specifically, we first perform two types of counterfactual evaluations to quantitatively study data valuation accuracy of LoGra on small-scale setups (Section 4.4.1). Then, we scale LoGra to LLMs and their massive training data, where we investigate qualitative accuracy (*i.e.*, how similar most valuable training data are to the model output) and memory/compute efficiency (Section 4.4.2). Finally, our appendix includes pseudo-code for LLM experiments (Appendix C.1) and experimental details such as hyperparameters and compute resources (Appendix C.2).

4.4.1 Quantitative Accuracy with Counterfactual Evaluation



(a) Brittleness test

(b) Linear datamodeling score (LDS)

Fig. 4.3 Quantitative accuracy evaluation of data valuation algorithms. We excluded TRAK in the WikiText experiments due to lack of a public implementation for language modeling tasks.

To quantitatively assess accuracy of data valuation algorithms, we adopt two counterfactual evaluation methods: brittleness test [77] and linear datamodeling score (LDS) [117]. First, the brittleness test focuses on accuracy in successfully identifying *top* valuable data. To this end, it first removes the top-k valuable data identified by each algorithm, retrains the model without them multiple times with different random seeds, and measures the overall change in the model output. The larger the output change is, the more accurate the algorithm is in identifying *top* valuable data. Second, LDS measures general valuation accuracy of *all* training data under the additivity assumption. Specifically, given multiple data subsets $\{S_i\}$ of the fixed size (*e.g.*, $|S_i| = |D|/2$), LDS estimates the test performance of the model trained on S_i by summing the values of all examples in S_i returned by each algorithm, and

compares it against the gold performance obtained by actually training the model on S_i using the Spearman correlation. Noting that linear datamodels have a connection to the game-theoretic data value (*e.g.*, Shapley value) [77], LDS can serve as a principled way to study data valuation accuracy.

We perform these counterfactual evaluations on three benchmarks where many rounds (up to 1800) of retraining is feasible: (1) MLP with FMNIST, (2) ResNet-9 [69] with CIFAR-10, and (3) GPT2 [126] with WikiText. On these benchmarks, we compare accuracy of LOGRA against four popular data valuation baselines, including gradient dot product [124], TRAK [117], EKFAC influence [55], and representation similarity [65]. With the aim of bearing relevance to a large-scale setting with LLMs and their vast training data, we have only considered baseline methods that satisfy the following two conditions. First, the method cannot retrain the model multiple times for identifying top-k valuable data.[1] Second, the method only has access to the final model checkpoint, which is the case for most LLMs. Given the above setup, we present our experiment results in Figure 4.3.

We observe that LOGRA slightly underperforms EKFAC influence, which is a few orders of magnitude slower in our large-scale experiments (Section 4.4.2), while noticeably outperforming other baselines. We attribute competitive accuracy of LOGRA to two factors. First, unlike TRAK of which projection dimension is limited by the huge projection matrix, LOGRA can efficiently afford a higher projection dimension thanks to its sublinear memory/compute costs for gradient projection, and thus achieve the higher expressivity. Second, gradient projection enables LOGRA to compute raw projected Fisher information matrix (or Hessian) without an approximation as in EKFAC influence. We expect that a more accurate computation of the Hessian generally leads to more accurate data valuation results.

Comparing the initialization schemes for LOGRA (PCA vs. random), we observe that LOGRA-PCA outperforms LOGRA-random on the FMNIST and CIFAR benchmarks. Hence, we hypothesize that it is generally more accurate to compute influence functions with larger components, similar to the spectral gradient sparsification effect of a damping term we discussed in Section 4.3.2. To understand a relatively poor performance of LOGRA-PCA on WikiText+GPT2, we point out that the Transformer architecture [158] used in this benchmark lacks the specialized KFAC Hessian approximation, unlike naive MLP [106] or convolutional [56] architectures in other benchmarks. Subsequently, our ad-hoc implementation of the PCA initialization based on the naive MLP architecture (*i.e.*, no weight sharing) may not successfully keep larger components of the GPT2 Hessian, failing to deliver its benefit. As a result, we decide to use LOGRA-random for our LLM experiments in the next subsection.

4.4.2 Scaling to Billion-Scale Models & Datasets

Given competitive accuracy of LOGRA, we now evaluate its practical utility in valuing billion-scale training data for billion-scale models. Specifically, we adopt GPT2-XL (1.5B) [126], Pythia-1.4B [16], and Llama3-8B-Instruct [2] as our models, and conduct data valuation on a random 1B-token subset of the OpenWebText (OWT) dataset [52]. The major motivations

[1] Note that multiple retraining is only allowed for evaluating accuracy of already identified top-k data, but not for identifying top-k data itself in our experiments.

behind choosing OWT as our data valuation dataset are twofold. First, we observe that OWT consists of relatively higher-quality data compared to other LLM training datasets like C4 [127] or Dolma [147] while maintaining the diversity unlike other high-quality datasets like WikiText [107]. Second, we anticipate that OWT largely overlaps with training datasets of all our models. In detail, GPT2-XL is trained on the WebText dataset that shares the same data curation process with OWT, Pythia-1.4B is trained on the Pile dataset [49] that includes an extension of OWT (*i.e.*, OpenWebText2), and we suppose a majority of OWT would be a part of Llama3's massive 15T-token pretraining dataset. We also note that our OWT subset size (*i.e.*, 1B tokens) was mainly limited by the available storage, not by compute (see Table 4.1). If we had access to a storage size of 1PB, performing data valuation with a dataset size of 100B+ tokens would be readily feasible using the same compute resource.

Efficiency. To begin with, we compare memory and compute efficiency of LoGra against EKFAC influence [55], the state-of-the-art and only algorithm that can run on billion-scale models without CUDA out-of-memory (OOM) errors. Indeed, we confirm that running TRAK or Arnoldi IF with billion-scale models results in CUDA OOM errors even on A100 GPUs with 80GB VRAM due to their gigantic projection matrix sizes. We report GPU memory usage and throughput of both logging (one-time) and influence computation (recurring) phases for the Llama3-8B-Instruct experiment with one A100 GPU and half-precision in Table 4.1.

	Logging (Compute & save Hessian|grad)				**Compute Influence** (Dot product between test & train grads)			
	Batch	Throughput	Memory	Storage	Train Batch	Test Batch	Throughput	Memory
EKFAC	1	1740 / 419*	71 / 80*GB	**89 GB**	4	4	12.2	75 GB
LoGra	1	3430	**23 GB**	3.5 TB	256	4	1599.6	**14 GB**
LoGra	16	**4696**	79 GB	3.5 TB	256	256	**79003.9**	**15 GB**

Table 4.1 Memory & compute efficiency analyses for LoGra and EKFAC. Throughput is measured as tokens/s for logging and (train, test) pairs/s for influence computations. * EKFAC logging consists of two subphases of KFAC fitting (left of /) and corrected eigenvalue fitting (right of /).

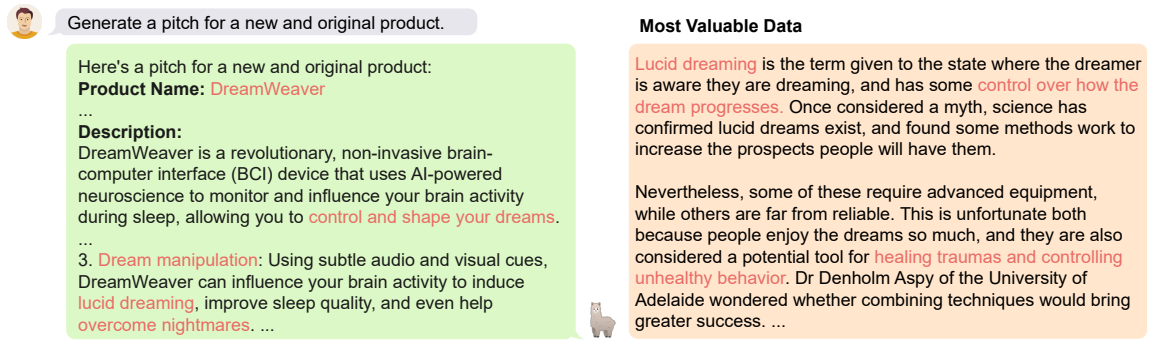
Due to the huge size of raw gradients (*e.g.*, 16GB in fp16 for an 8B model), EKFAC cannot afford storing raw gradients for *all* training data to disk. As a result, EKFAC needs to recompute all training gradients for each test batch, and thus requires allocating extra GPU memory on model weights and intermediate activations. This largely limits both train/test batch sizes and throughput (12.2 pairs/s), and performing data valuation with EKFAC for 256 test data and 1B-token training data would take 11,300 A100 GPU hours, rendering it hardly usable in most practical setups.

In contrast, with its (efficient) gradient projection, LoGra not only significantly improves compute and memory efficiency, but also avoids training gradient recomputations at the costs of disk space for storing *projected* training gradients and latency from data IO. Since the storage cost is typically much cheaper than the compute cost[2], we believe our trade-off offers considerable practical benefits. Furthermore, we can largely hide the data

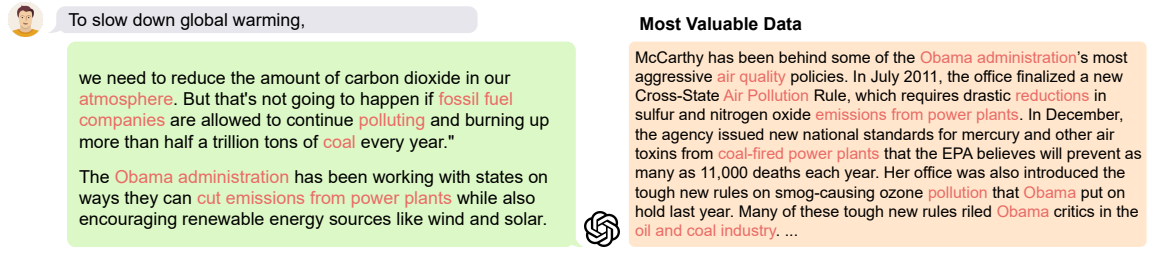
[2] *e.g.*, hourly rates for a 1TB storage and one A100 GPU are approximately $0.03 and $4 on AWS.

IO cost by overlapping gradient reading/writing processes with other computations. For instance, given the fixed train gradient batch size of 256 (*i.e.*, fixed data loading time), we are able to successfully overlap the process of loading training gradients from disk with influence computations against up to 256 test gradients, and thereby achieve almost 6,500× improvement in throughput from EKFAC influence. Noting that our GPU memory usage is far from saturated even with the train/test batch size of 256, we believe that more throughput improvements can be achieved simply by further increasing train/test batch sizes.

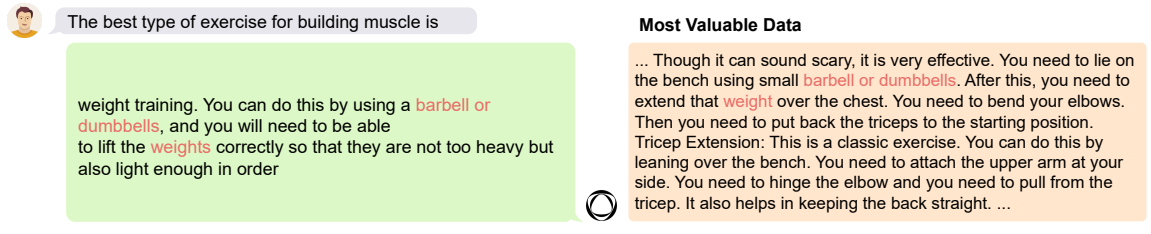
Qualitative Accuracy. Next, we analyze qualitative similarities between queried LLM outputs and most valuable data identified by LOGRA that can be critical for promoting trust in the data valuation system [167]. Importantly, we observe that naive influence functions frequently return outlier data with high gradient norms as most valuable data, as also noted in [12, 55]. To mitigate this issue, we instead use *l*-RelatIF, a variant of influence functions that normalizes the original influence score with the self-influence score of each training data to penalize such outlier effects [12]. Our experimental results are provided below.



(a) Llama3-8B-Instruct



(b) GPT2-XL (1.5B)



(c) Pythia-1.4B

Fig. 4.4 Qualitative accuracy of data valuations with LOGRA. Important keywords in each example are *manually* highlighted for the improved readability.

Llama3-8B-Instruct Results

Prompt

To improve data privacy,

Response

To improve data privacy, consider the following strategies:
1. **Implement robust data encryption**: Use end-to-end encryption to protect data in transit and at rest. This ensures that even if an unauthorized party gains access to the data, they won't be able to read or understand it.
2. **Use secure protocols**: Use secure communication protocols, such as HTTPS, SFTP, and SSH, to protect data during transmission.
3. **Limit data collection**: Only collect the minimum amount of data necessary to achieve your goals. Avoid collecting sensitive data unless it's absolutely necessary.
4. **Pseudonymize or anonymize data**: Consider pseudonymizing or anonymizing data to protect individual privacy while still allowing for data analysis and research.
5. **Implement access controls**: Establish strict access controls to ensure that only authorized personnel can access sensitive data. This includes multi-factor authentication, role-based access control, and data masking.
6. **Monitor and audit data access**: Regularly monitor and audit data access to detect and respond to potential data breaches.
7. **Use data minimization techniques**: Implement data minimization techniques, such as data aggregation, to reduce the amount of sensitive data stored and processed.
8. **Conduct regular security assessments**: Conduct regular security assessments and penetration

First Most Valuable Data

American public, overseers and stakeholders. (3) Improve civil liberties and privacy protections by supporting research, education and training. (4)Advise NSA leaders on civil liberties and privacy issues.

In this new report, the CLPO uses the framework Fair Information Practice Principles (FIPPs) to evaluate how the NSA safeguards general civil liberties and privacy. Fair Information Practice Principles FIPP's are broken down in eight founding parts for federal agencies to adhere to. These principles are:

Transparency: organizations should be transparent and notify individuals regarding collection, use, dissemination, and maintenance of personally identifiable information (PII)
organizations should be transparent and notify individuals regarding collection, use, dissemination, and maintenance of personally identifiable information (PII) Individual participation: (Skip).
organizations should involve the individual in the process of using PII. (Skip)
organizations should specifically articulate the authority that permits the collection of PII and specifically articulate the purpose(s) for which it will be used Data minimizations: (Skip)
organizations should only collect PII that is directly relevant and necessary to accomplish the specified purose and only retain PII for as long as it is necessary to fulfill the specified purpose. (Skip)
Organizations should use PII solely for the purpose specified in the notice. (Skip)
Organizations should protect PII (in all media) though appropriate (Skip)

Second Most Valuable Data

(Skip) Easy reporting with out-of-the-box compliance reports – To achieve GDPR compliance, organizations need to demonstrate that they monitor critical infrastructure holding personal data of EU citizens.

Using a powerful compliance and forensics reporting engine, Exabeam generates a series of built-in, GDPR-specific reports, that help reduce the time to prove compliance to auditors.

Protect employee Personally Identifiable Information (PII) – A critical GDPR requirement is to protect employee PII from unwarranted access. Exabeam provides role-based access control (RBAC) that can be used to enforce PII data masking. (Skip)

Fig. 4.5 Example 1. Llama3-8B-Instruct data valuation result.

Prompt

How can one become a billionaire?

Response

Becoming a billionaire requires a combination of hard work, smart financial decisions, and a bit of luck. Here are some strategies that can increase your chances of achieving this goal:

1. **Start with a great business idea**: Identify a market need and create a product or service that solves a problem or meets a demand. Be innovative and stay ahead of the competition.
2. **Work hard and be persistent**: Building a successful business takes time, effort, and perseverance. Be willing to put in the long hours and make sacrifices to achieve your goals.
3. **Develop a strong entrepreneurial mindset**: Be willing to take calculated risks, adapt to changing circumstances, and continuously learn and improve.
4. **Build a strong team**: Surround yourself with talented and dedicated individuals who share your vision and values.
5. **Make smart financial decisions**: Invest wisely, manage your finances effectively, and avoid debt.
6. **Diversify your investments**: Spread your investments across different asset classes, such as stocks, real estate, and bonds, to minimize risk.
7. **Be patient and disciplined**: Building wealth takes time, so be patient and disciplined in your financial decisions.
8. **Stay informed and educated**1: Continuously educate yourself on business, finance, and investing to make informed

First Most Valuable Data

(Skip) Make the most money. Demand for WordPress development is high and clients are willing to pay more for developers who are the best in their field.
. Demand for WordPress development is high and clients are willing to pay more for developers who are the best in their field. Get the best clients. When you are at the top, you have the freedom to say “No” to the projects you don’t want, and “Yes” to the projects you do.
. When you are at the top, you have the freedom to say “No” to the projects you don’t want, and “Yes” to the projects you do. Have the most influence. Being at the top means you have influence (and responsibility) and the capability to shape the future of WordPress as well as the ecosystem that is built around it.
One Hour of Reading a Day
If you’re going to make it to the top, then you need to spend at least one hour each workday focused on reading and learning more about WordPress — outside of any development work. (Skip)

Second Most Valuable Data

(Skip) A hugely successful game is more about psychology than merely some lovely design or cool coding.
As a successful gaming company, you need to:
Take time to understand Human Psychology
Give before you receive
Be patient and persistent, but not offensive, in your efforts to sell
Apply. Learn. Revise. Repeat.(Skip)

Third Most Valuable Data

(Skip) Unfortunately, many companies attempt to become agile by implementing only the process part and ignoring the fundamental change in people management that leads to true harmony. I think we've all been there.
I don't claim to be an expert in this area, but here are some pointers that have helped me build harmonious teams in the past.
1: Be careful when shuffling team members. (Skip)
2: Control negativity by reframing the problem. (Skip)
3: Praise in public, have difficult conversations in private. (Skip)
4: Rephrase criticisms into a request for help. (Skip)
5: Protect your team at all costs. (Skip)
6: Turn problems into action items. (Skip)

Fig. 4.6 Example 2. Llama3-8B-Instruct data valuation result.

Prompt

Do aliens exist?

Response

The existence of aliens, or extraterrestrial life, is a topic of ongoing debate and research in the fields of astrobiology, astrophysics, and the search for extraterrestrial intelligence (SETI). While there is currently no definitive evidence of alien life, there are many reasons to believe that the possibility of life existing elsewhere in the universe is quite high.

Here are some of the key arguments for and against the existence of aliens:

Arguments for the existence of aliens:
1. **The vastness of the universe**: With an estimated 100-400 billion stars in the Milky Way galaxy alone, and over 100 billion galaxies in the observable universe, the potential for life-supporting planets is immense.
2. **The discovery of exoplanets**: Over 4,000 exoplanets have been discovered so far, and many of these planets are believed to be located in the habitable zones of their respective stars, where conditions are suitable for life as we know it.
3. **The building blocks of life**: Many organic compounds, such as amino acids and sugars, have been found in meteorites and in interstellar space, suggesting that the raw materials for life are widespread in the universe.
4. **The existence of extremophiles**: On Earth, there informed

First Most Valuable Data

(Skip) Prior to Seticon II in 2012 Tarter addressed concerns about alien invasion, like those seen in science fiction films, in particular those expressed by Stephen Hawking. Correlated with Hawking's reservations is his belief that humans should not attempt to broadcast signals in order to contact alien civilizations (Active SETI). Tarter thinks that aliens evolved enough to communicate with and travel to Earth will be long-lived enough to have needed to grow beyond violent tendencies.[25] [26]

“Often the aliens of science fiction say more about us than they do about themselves.... While Sir Stephen Hawking warned that alien life might try to conquer or colonize Earth, I respectfully disagree. If aliens were able to visit Earth that would mean they would have technological capabilities sophisticated enough not to need slaves, food, or other planets.(Skip)

Second Most Valuable Data

(Skip) …Or Are The Aliens Just Hiding?
There is another class of possible solutions to the Fermi paradox that boil down to the fact that alien civilisations do exist, but we simply see no evidence of them.
Distance scales: Perhaps civilisations are spread too thinly throughout the Galaxy to effectively communicate with each other? Civilisations may be separated in space, and also in time, so two civilisations just don't overlap during the time that they're each active.
Technical problems: Maybe we're not looking in the right place, or in the right way? Or maybe we just haven't been looking for long enough? Perhaps we've not recognised a signal that's out there, because the alien civilisation is using technology that we simply cannot comprehend.
Alcamo Marina - Torre Alkamar in zona Battigia Wikimedia Commons Isolationist: Perhaps the aliens are out there, but they're choosing to hide themselves from us? Perhaps everyone is listening, but nobody is transmitting? It may be that other civilisations know we're here, but the Earth is purposely isolated, as if we're some kind of exhibit in a zoo. (Skip)

Third Most Valuable Data

(Skip) About aliens, Li claimed that extraterrestrial aliens are actively intervening in human affairs.[66][67] Li claimed that aliens developed and introduced the technology used by humans today.[68][69] Li has denounced modern technology as part of an alien plot against humanity.[70] Li believes humans are being impersonated by alien agents.[71] <|begin_of_text|>Contractor Safedem used 'good practice' but 'underestimated the robustness' of the buildings.(Skip)

Fig. 4.7 Example 3. Llama3-8B-Instruct data valuation result.

Prompt

What is the difference between machine learning and deep learning?

Response

Machine learning and deep learning are both subfields of artificial intelligence (AI) that involve training algorithms to perform specific tasks. The key difference between them lies in the complexity and depth of the algorithms used.

Machine Learning (ML):
Machine learning is a type of AI that enables computers to learn from data without being explicitly programmed. It involves training algorithms on labeled data to recognize patterns and make predictions or decisions. ML algorithms can be categorized into three types:
1. Supervised Learning: The algorithm learns from labeled data to predict outcomes.
2. Unsupervised Learning: The algorithm discovers patterns and relationships in unlabeled data.
3. Reinforcement Learning: The algorithm learns through trial and error by interacting with an environment.

Machine learning algorithms are typically shallow, meaning they have a limited number of layers (usually 1-2) and are designed to solve specific, well-defined problems. Examples of machine learning applications include:
* Image classification (e.g., recognizing objects in images)
* Natural Language Processing (NLP) for text classification or sentiment analysis
* Recommendation systems for personalized product suggestions

Deep Learning (DL):
Deep learning is a subfield of machine learning that involves the use of neural networks with multiple layers

First Most Valuable Data

(Skip) three types mainly differ from one other in the they use to train the machine.
In Supervised Learning, the machine is trained using a certain dataset. (Skip)
In Unsupervised Learning, the machine is not trained using an initial dataset but rather it has only a function which accepts certain arguments and after performing certain computations, an outcome is generated. (Skip)
In Reinforcement Learning, the machine is yet again not trained on any initial dataset but then as the programme executes, it learns, formats its own mistakes and improves itself.(Skip)
Like I mentioned before, Artificial Intelligence and Machine learning go hand in hand.(Skip)

Second Most Valuable Data

(Skip) In supervised learning, the learning itself is what you care about. You've got your cost function, which you want to minimize. In unsupervised learning, the goal is always to help some other task, like classification or categorization. For example, I might ask a computer system to passively watch a lot of YouTube videos (so unsupervised learning happens here), then ask it to recognize objects with great accuracy (that's the final supervised learning task). (Skip)
DB: What are some other areas where you see exciting progress?
IS: A general direction that I believe to be extremely important is: are learning models capable of more sequential computations? I mentioned how I think that deep learning is successful because it can do more sequential computations than previous ("shallow") models. (Skip)

Third Most Valuable Data

(Skip) Eleni: Can you clarify the difference between AI, machine learning, and deep learning?
Inga: AI is the umbrella over machine learning. AI, to me, is our hope for human intelligence exhibited by machines. There are multiple goals of AI, which are all facets of human intelligence. We can reason, accumulate knowledge, plan for things, manipulate objects, and communicate with each other with our language. Learning, of course, is a part of what makes us human.
Machine learning is the ability to learn without being explicitly programmed. The reason why machine learning has become so popular in recent years is that teaching a machine to learn—or building a machine that can learn on its own—can achieve all the goals of AI. It can learn how to reason, and understand our language, and perceive and move objects. You give a machine learning algorithm lots and lots of data and that algorithm learns the concepts around this data. It's then able to make a determination or prediction about something in the world.
Deep learning is the newest field of machine learning, and it has really catapulted us into the renaissance that we're in today. The "deep" in deep learning comes from multiple hidden layers of transformation in data. Examples of things that are enabled by deep learning are self-driving cars or the Google Translate app in which the camera on your phone can immediately translate text that it sees into a different language.(Skip)

Fig. 4.8 Example 4. Llama3-8B-Instruct data valuation result.

Prompt

Create a comic strip illustrating the life of a software engineer.

Response

Here is a comic strip illustrating the life of a software engineer:
Panel 1: Morning Routine
[Image description: A software engineer, John, sitting at his desk with a cup of coffee and a laptop. He's wearing a t-shirt and jeans, and has a messy hair.]
John: (thought bubble) Ugh, another day, another bug to squash.
Panel 2: Code Review
[Image description: John sitting at his desk, staring at his laptop screen with a puzzled expression. His manager, Sarah, is standing behind him, pointing at the screen.]
Sarah: "John, this code is a mess. Can you explain what you're trying to do here?"
John: "Uh... um... I was trying to... uh..."
Panel 3: Meeting Mayhem
[Image description: John sitting in a conference room with his team, surrounded by whiteboards and sticky notes. The team lead, Mark, is talking enthusiastically.]
Mark: "And then, we'll integrate the new API with the existing framework, and—"
John: (thought bubble) Zzz... I've heard this before...
Panel 4: Lunch Break
[Image description: John walking to the cafeteria, carrying a sandwich and a book.

First Most Valuable Data

(Skip) Pyrrha woke up a little after dawn. She got up, made a quick stop in the bathroom, then wandered into the lounge. Summer was sitting on one of the couches, a magazine in her hands and a steaming cup of coffee on the table before her.
"Morning Pyrrha." She said.
"Good morning Summer." Pyrrha yawned. She took a seat on the couch opposite Summer and stretched out.
"Pyrrha, is everything alright?" Summer asked.
"Yes, why?" Pyrrha asked.
"Did you have a bad dream last night?" Summer asked. "I heard some whimpering…"
"Oh...uh...I don't know." Pyrrha lied. "I usually don't remember my dreams." Summer shrugged and went back to her magazine.
After a few minutes Pyrrha got up and made her way to the kitchen. She put some bread in the toaster and started making some coffee. As the freshly made toast popped up Yang walked in with a yawn and a stretch. "Hey Pyr." Yang said. "Smells good."
"Good morning Yang." Pyrrha said. "Sleep well?" (Skip)

Second Most Valuable Data

(Skip) Ruby cheered. "I wish you both could have been there."
"This is your best work for sure." Blake chuckled. "I don't suppose you take commissions?"
Ruby laughed. "I can be persuaded to."
(Skip)
As she had after the previous class, Ruby packed up and rushed home once it was over. She was one of the last to leave, working late to finish up her sketch of Pyrrha as a Mistrali hoplite. Few students wandered the paths this late, but she was not interested meeting up with anyone anyway. Ruby just wanted to get back to her room and rest after a long night.
Walking fast she turned a corner around a small maintenance shack and nearly ran into a girl standing in the middle of the path.
"Wah...sorry!" Ruby gasped. The girl, mint-haired and olive-skinned, just stared back. "Um…"
"Ruby Rose." The girl said flatly.
"Uh...yeah." Ruby confirmed. "Who are you?" The girl shifted her position, closing on Ruby while simultaneously forcing her toward the wall of the shack. Ruby backed up nervously but soon bumped into the wall.
The girl leapt forward and grabbed her by the shoulders. "You didn't think you'd get away with that little stunt did you?" The girl hissed.(Skip)

Fig. 4.9 Example 5. Llama3-8B-Instruct data valuation result. LoGra identifies novel literature as most valuable data.

Prompt

Compose a tweet that addresses the issue of environmental degradation.

Response

"The clock is ticking! Our planet is crying out for help as environmental degradation worsens. Let's take action NOW to reduce plastic waste, conserve water, and protect our natural habitats. The future of our planet depends on it #GoGreen #Sustainability #EnvironmentalProtection"

First Most Valuable Data

Learn more about the winner and the contest results (good news: behavior changed for the better!) in our recap blog post and this one-page summary
Download. Drive better. Win.
Up for a little friendly competition and safer streets?
Seattle's Safest Driver is an app-based safe driving competition, complete with cash prizes and bragging rights. In partnership with PEMCO Insurance and Cambridge Mobile Telematics, the 2-month competition seeks to change behavior and support Seattle's Vision Zero goal of ending traffic deaths and serious injuries on city streets by 2030.
Step 1 - Get the app and get moving.
Please note, the app is no longer available.
Download the free app and it'll start tracking your trips - by car, bike, and transit (ferry, too!). You get scored for trips when you're behind the wheel. The app measures speed, acceleration, braking, cornering, and phone distraction. The safer you are, the higher your score, and the more chances you have to win a variety of prizes (mmm hmm, prizes - read more about them in Step 3).
Anyone can download the app, but we do have eligibility requirements for prizes. You need to be at least 18 years old, a resident of King, Pierce, Snohomish, or Kitsap County, and a portion of your trips must occur within the city of Seattle. This is, after all, a competition for Seattle's safest driver. For all the details, check out the contest's terms and conditions.
Step 2 - Challenge your friends, family, and coworkers. Then beat them.
Settle the age-old debate of who's the better driver - invite your friends and family to compete and see how you rank against each other. Get feedback and tips on how to improve, and earn achievement badges. Friendly competition is fun, but traffic safety is no game. Working together, we can make Seattle's streets safer for everyone.
Step 3 - Compete for prizes!
We realize it takes some carrots to incentivize good behavior. Thanks to a partnership with PEMCO Insurance - a company that shares our vision for safe streets - eligible participants can compete for nearly $15,000 in prizes, including a grand prize of $2,000. Cash. 2nd place gets $1,000, and 3rd place gets $500.

Second Most Valuable Data

(Skip) “I get very hangry if I miss a meal.”
The most important thing in my food plan, is that I actually eat. For many years I didn’t get hunger pains, I got nausea. Which, I recently learned, is quite common. I never “wanted” to eat. Too nauseous, tired, and pained to cook anything, I essentially lived on coffee and cookies during the day until about 5 years ago.
These tips assume you’re already on your chosen type diet, if you haven’t found one yet there’s a treasure trove of condition-specific advice available on the internet.
Tip #1
Remember why you’re doing an elimination type diet in the first place.
Despite having so many things wrong with me, I never would have agreed that food would make a difference. If someone said, “You should try eating better.” I would have thrown a shoe at them. I have an imbalance in my brain, it’s not a food problem! Or, is it?
My new relationship with food started out of necessity. After a bad go with narcolepsy meds, I found myself unable to take big pharm meds anymore. Everything gave me terrible side effects, even over the counter meds. I wasn’t quite sure what to do and I stumbled on a TedX talk by Dr. Terry Wahls. She had pushed her MS (multiple sclerosis) into remission with just food. She did a lot of research on her own on pub med, in hopes of getting out of the zero gravity wheelchair she’d ended up in, despite the best doctors and treatments. (Skip)

Fig. 4.10 Example 6. Llama3-8B-Instruct data valuation result (failure).

GPT2-XL Results

Prompt

To measure economy inflation,

Response

the Fed uses a price index called CPI. The Bureau of Labor Statistics (BLS) calculates this number by adding up all items in its Consumer Price Index and dividing it by total employment to get an estimate for what consumers are paying on average each month.

The BLS also publishes another statistic that measures how much money is being spent per worker: Gross Domestic Product or GDP. This figure includes everything from consumer spending at stores like Walmart to government expenditures such as Social Security payments. It's calculated using data collected through surveys conducted with households about their income and expenses.

First Most Valuable Data

(Skip) According to the BLS, the average U.S. worker earned $22.87 an hour during the month of March, which is exactly the same number we saw in February. So inflation is going up and wages are staying flat. That means that American family budgets are going to be squeezed even more. In addition, the numbers from the BLS show that it is still incredibly difficult to get a job. In fact, the average length of unemployment in the U.S. is now an all-time record 39 weeks. (Skip)

Second Most Valuable Data

(Skip) The Social Security Administration bases the COLA on a measure of inflation called CPI-W, a statistic that captures how fast costs are rising for workers. Most seniors are retirees who no longer have jobs. Health care is their biggest expense, and it’s one of the fastest rising costs in America. (Skip)

Third Most Valuable Data

(Skip) Despite the positive spin in the advanced 1st Q GDP results for consumer consumption (+2.2% annualized gain) earlier in the week, yesterdayÃ¢â‚¬â„¢s data showed that US consumption fell off in Mar. even after the strong gains in the 1st 2-months of this year (-0.2% vs. -0.1%, m/m). Analysts believe the report suggests that the consumer Ã¢â‚¬Ëœmay have received a temporary boost from lower tax payments and government transfersÃ¢â‚¬â„¢. ItÃ¢â‚¬â„¢s worth noting that with personal income falling (-0.3% vs. -0.2%) for 6 out of the past 7-months, consumption can be expected to remain weak in the foreseeable future. With government transfers and lower tax payments supporting growth in Jan. (+1.1%) and Feb. (+0.4%), an even faster pace of deterioration in salaries in Mar., along with weaker proprietors’ income (-0.6% m/m), rental income (-3.7%) and supplements to wages & salaries (-0.1%), have all contributed to this weaker headline. The personal savings rate advanced last month to +4.2%, all on the back of job and economic uncertainty, by default this was bound to affect consumption. With the Fed preferring to use the PCE deflator as their inflation measure of choice as opposed to us using CPI, last monthÃ¢â‚¬â„¢s deflator eased to +0.6%, y/y, further evidence that this is the lowest growth rate in 50-years. With the y/y rate at +1.8% the Fed should be concerned about disinflationary pressures! (Skip)

Fourth Most Valuable Data

(Skip) price inflation, which is a rise in the general level of prices of goods and services in an economy over a period of time.[65] A consumer price index (CPI) is a measure estimating the average price of consumer goods and services purchased by households. The United States Consumer Price Index, published by the Bureau of Labor Statistics, is a measure estimating the average price of consumer goods and services in the United States.[66] It reflects inflation as experienced by consumers in their day-to-day living expenses.[67] A graph showing the U.S. CPI relative to 1982–1984 and the annual year-over-year change in CPI is shown at right. (Skip)

Fig. 4.11 Example 1. GPT2-XL data valuation result.

Prompt

Dogs make better pets than cats because

Response

they are more social and have a higher tolerance for stress.

Cats, on the other hand, can be very destructive to their environment if left alone too long or not exercised enough. They also tend to get into fights with each other over food which is why it's important that you keep them in an area where there isn't much competition between your two dogs!

First Most Valuable Data

(Skip) 'Social sensitivity appears to play an important role in pet and shelter dogs' willingness to engage in problem-solving behaviour, which could suggest generalized dependence on, or deference to, human action,' she said. 'While an increased proclivity for looking at humans may represent a cognitive shift in dogs compared with wolves, it does not necessarily suggest cognitive advancement. Dogs heightened social response may interfere with their problem-solving behaviour (Skip)

Second Most Valuable Data

(Skip) 3. Bengals tend to be fairly high maintenance

Bengals do as a rule tend to be demanding and fairly high maintenance cats. They often bond strongly with their families and can be very demanding of their time and attention, which is just what many people want-but of course, this is not for everyone, and can drive some owners mad! If you don’t have time to devote to a Bengal every day,then it may not be the cat for you.

Owning a Bengal can in many ways be more like owning a dog than a cat, as they will often follow you around the house, yell for attention and want to get involved in everything that you are doing-and they very much dislike being ignored!

4. They can be noisy

Before you welcome a Bengal cat to your family, you should (Skip)

Third Most Valuable Data

(Skip) Temperament [edit]

Siamese are usually very affectionate and intelligent cats, renowned for their social nature. Many enjoy being with people and are sometimes described as "extroverts". Often they bond strongly to a single person. Some Siamese are extremely vocal, with a loud, low-pitched voice—known as "Meezer", from which they get one of their nicknames[6]—that has been compared to the cries of a human baby, and persistent in demanding attention. These cats are typically active and playful, even as adults, and are often described as more dog-like in behavior than other cats.[14][15] (Skip)

Fig. 4.12 Example 2. GPT2-XL data valuation result.

Prompt

Will AI replace humans?

Response

The question of whether or not machines will eventually be able to outperform human beings in all areas is a hot topic. The answer, according to some experts and futurists, could come as soon as 2040 — but it's still too early for that prediction to have any real meaning at this point.

First Most Valuable Data

(Skip) Let’s not place humanity in the position of those ants. Please encourage your students to think not only about how to create AI, but also about how to ensure its beneficial use.
John Leonard:
TL;DR: Not any time soon: There’s a lot of merit to what [Hawking] is saying, but I think the question is: what’s the timeline? Is this going to unfold over years, decades or centuries? In the field, there’s this feeling of exponential growth. But as a roboticist trying to get robots to do things like drive on streets and walk up and down stairs, I can see clearly that there are whole other parts of the problem that remain unsolved. I think this notion of evil A.I., if it happens at all, is many, many years in the future—and I think we have much greater societal challenges to worry about, in the here-and-now. My view, as a roboticist, is that trying to get robots to do things like drive safely in urban traffic, or make left turns, or make real decisions within the wall of information—these are problems that were hard 30 years ago, and they’re still hard today. I would claim our advances in a more broad A.I. are actually pretty lame; we’re not making as much progress as some people might say.
Will Robots Take Our Jobs?
TL;DR: If by robots, you mean Amazon If machines produce everything we need, the outcome will depend on how things are distributed. Everyone can enjoy a life of luxurious leisure if the machine-produced wealth is shared, or most people can end up miserably poor if the machine-owners successfully lobby against wealth redistribution. So far, the trend seems to be toward the second option, with technology driving ever-increasing inequality.
Leonard:
TL;DR: Again, not any time soon (Skip)

Second Most Valuable Data

(Skip) How soon, and should we welcome or fear it?
The first big controversy, dividing even leading AI researchers, involves forecasting what will happen. When, if ever, will AI outperform humans at all intellectual tasks, and will it be a good thing?

Do you want superintelligence?
Everything we love about civilization is arguably the product of intelligence, so we can potentially do even better by amplifying human intelligence with machine intelligence. But some worry that superintelligent machines would end up controlling us and wonder whether their goals would be aligned with ours. Do you want there to be superintelligent AI, i.e., general intelligence far beyond human level?

What Should the Future Look Like?
In his book, Tegmark argues that we shouldn’t passively ask “what will happen?” as if the future is predetermined, but instead ask what we want to happen and then try to create that future. What sort of future do you want?

If superintelligence arrives, who should be in control?
If you one day get an AI helper, do you want it to be conscious, i.e., to have subjective experience (as opposed to being like a zombie which can at best pretend to be conscious)?(Skip)

Fig. 4.13 Example 3. GPT2-XL data valuation result.

Prompt

Interventions that may reduce drug abuse include:

Response

• Encouraging parents to talk with their children about drugs and alcohol. Parents should be encouraged not only to discuss the dangers of using illegal substances, but also how they can help prevent them from happening in the first place (e.g., by keeping a close eye on what is going into or coming out of their home). • Providing information regarding safe use of prescription medications such as pain relievers and anti-anxiety medication; providing education for health care providers who treat patients suffering from substance dependence. This includes educating physicians about prescribing opioids safely when prescribed appropriately. • Educating students at school about the risks associated with illicitly obtained narcotics. Students are more likely than adults to engage in risky behaviors if given access to these products through friends or family members. In addition, many adolescents have no idea where to obtain heroin because it has been replaced by synthetic versions made without oversight by pharmaceutical companies. The availability of this product increases risk factors including addiction, overdose death, criminal activity, and HIV transmission among injection users. • Increasing awareness within communities surrounding the need to address issues related to youth involvement in street crime. These efforts must focus specifically on addressing the needs of young people involved in gangs rather than focusing solely upon those individuals engaged in other forms of violence.

First Most Valuable Data

(Skip) addictive and have horrific side effects."
The measure was sent to the governor on a 35-21 vote. Quinn has said he was "open minded" to the proposal after meeting with a veteran who used marijuana to ease pain related to war wounds. On Friday, his office said that he will "carefully review" the legislation when it reaches his desk.
The Democratic governor must weigh not only the merits of the bill but also the politics, given his re-election campaign next year. To a large extent, Quinn has positioned himself as a liberal, signing off on a major income tax increase, approving civil unions and abolishing the death penalty while in office.
Critics of the marijuana legislation argued that while the intentions may be good, the bill would encourage use of a gateway drug that could lead users to harder substances, destroying families along the way. (Skip)

Second Most Valuable Data

(Skip) who is 19 years of age or older [in Ontario], but we do expect that students do not show up to the school in possession of alcohol or having consumed alcohol."
The Ontario government's health and physical education curriculum already teaches students about substance use and addiction. The Durham board runs a "healthy choices" program, which focuses on the risks associated with alcohol and other drugs, whether legal or illegal.
Legalization likely won't "dramatically" change the conversation with students about marijuana, Bowyer said, because the impact and possible consequences of using it remain the same. (Skip)

Third Most Valuable Data

not for human consumption" or "not for sale to minors." The clinical effects of most new designer drugs can be described as either hallucinogenic, stimulant, or opioid-like. They may also have a combination of these effects due to designer side-chain substitutions. The easy accessibility and rapid emergence of new designer drugs have created challenges for health care providers when treating patients presenting with acute toxicity from these substances, many of which can produce significant and/or life-threatening adverse effects. Moreover, the health care provider has no way to verify the contents and/or potency of the agent ingested because it can vary between packages and distributors. Therefore, a thorough knowledge of the available designer drugs, common signs and symptoms of toxicity associated with these agents, and potential effective treatment modalities are essential to appropriately manage these patients. (Skip)

Fig. 4.14 Example 4. GPT2-XL data valuation result.

Prompt

Can humans fall in love with robots?.

Response

The answer is yes. But it's not a romantic relationship, and the robot doesn't have to be human-like or even sentient for that matter — just as we don't need our pets to look like us (or at least act like them). The question of whether people will want their own personal AI companions has been debated since before computers were invented; some argue they'll never happen because there are too many other things on Earth worth living for. Others say this isn't so much about what you can do but rather how well your life could work out if someone else was around all day instead of being left alone by themselves. And then there are those who think having an intelligent companion would make one more fulfilled than ever: "I'm sure I'd feel happier knowing my partner had something better going," says Dr. David Levy from University College London. He believes that while he wouldn't necessarily choose his spouse over another person, he might prefer him/her to live longer.

First Most Valuable Data

(Skip) “This is of critical importance and perhaps one of the biggest factors eroding campaign finance contribution limits,” said Craig Holman with Public Citizen. “Many single candidate super PACs are probably set up by the candidate’s own staff or supporters or family for that matter, so these are means for candidates and political parties to get around the limits, and they can take unlimited amounts of contributions.”
Last election, former Florida Gov. Jeb Bush (R) was able to solicit nearly $100 million for the super PAC Right to Rise, because he did so before officially announcing he was running for president, working around laws prohibiting coordination. The earlier it was formed, and the longer he put off his declaration of candidacy, the longer the super PAC could work with Bush’s team and fill the group’s coffers.
We haven’t found any such blatant ties among this year’s crop of super PACs, but there are some familiar names. Main Voters’ treasurer is Seth Tanner, an alum of the teams of Sen. Elizabeth Warren (D-Mass), former Gov. Bill Richardson (D-N.M.), and its custodian of records is Amy Pritchard, a political strategist and DNC alum. America First Action, Inc. lists Charles Gantt as the custodian of records, who was the Chief Financial Officer of Trump for America, Inc. Lab 736’s treasurer, Kate Gage, is a former Obama policy adviser. (Skip)

Second Most Valuable Data

(Skip) ArbCom should certainly examine how its procedures can contribute to harassment. The length of time involved is a major problem. An outside advisor would help. Smallbones (talk) 03:25, 21 June 2016 (UTC)
This would have to be handled carefully, but I think that an outside advisor would be a great idea. Most large and powerful committees should have someone to serve as a separate witness/observer. That person might not have the power to actually stop something, but it would be beneficial to have someone who could serve in this position and raise valid points - as well as being a good person that the committee could turn to for any questions they might have as well. Tokyogirl79 (talk) 03:41, 21 June 2016 (UTC)
I wouldn't say "adult supervision," but some kind of expert advisor on things like conflict resolution, group dynamics and ethics sounds like a good deal. I would say an eighteen-month trial period that automatically expires unless renewed. If it turns out that having this person around prevents the Arbitrators from doing their jobs, off s/he goes. Darkfrog24 (talk) 17:41, 21 June 2016 (UTC) One problem I've noticed among long-serving admins is that they become very immersed in Wikipedia's rules, to the point where they forget how unintuitive they can be to newbs and even longstanding editors who've focused on other areas. An outsider would give much-needed perspective. Darkfrog24 (talk) 03:14, 22 June 2016 (UTC) (Skip)

Third Most Valuable Data

(Skip) BB: OK, so you have statistics to back it up, but it still seems like it's a risky proposition. What's in it for the employer to reduce hours?
MB: That's the thing. I take a lot of credit for this, but it's my brother and my mother who actually started the company and started with six-hour workdays. But they worried that for the company, that you wouldn't get as much done. But we do. And the benefits are that we have happy staff. And we can hire the best staff. People come to us looking for jobs every week. And when we find good staff, we want to keep them because the staff is the most important thing we have. Some of our employees have been offered other jobs with more salary, and they didn't take the job.
BB: So you have an advantage over your competitors then. But wouldn't that advantage disappear if all of your competitors adopted a 30-hour work week as well?
MB: Of course. So let's not do that [laughs]. (Skip)

Fig. 4.15 Example 5. GPT2-XL data valuation result (failure).

Pythia-1.4b Results

Prompt

Cristiano Ronaldo is

Response

the best player in world football.
The Portuguese superstar has been named FIFA's World Player of The Year for a second time, after winning his third Ballon d'Or award on Sunday night (18 February).

First Most Valuable Data

(Skip) Just ask the U.S. Women's Soccer Team.

This stuff isn't just about statistics and averages. There are many concrete and specific examples of the ways in which the pay gap is a real force in women's lives today. (See hospital doctors, Jennifer Lawrence and even Buffer, a company obsessed with eliminating the pay gap that took a hard look at itself and found...a pay gap.) But while it's certainly disheartening that five members of the U.S. women's national soccer team had to file suit against the U.S. Soccer Federation to push for the same pay and bonuses as their male counterparts, it also makes for kind of a perfect, high-profile example of the pay gap in action just in time for Equal Pay Day. (Skip)

Second Most Valuable Data

Barcelona finish 2nd in the league promising them a Champions League spot.

Ronaldinho Gaucho Biography stats show that the following year at Barcelona would be perhaps the best year in Ronaldinhos career winning the league title in 2004-05 and being named the FIFA World Player of The Year in 2004.

Ronaldinho took the FIFA World Player of The Year once again the following year, beating Chelsea's Frank Lampard and Barcelona teammate Samuel Eto'o to become 2 consecutive time FIFA World Player of The Year. (Skip)

Third Most Valuable Data

(Skip) • Best Championship Performance: Kawhi Leonard, who led the Spurs past Miami in the NBA Finals, averaging 17.8 points and 6.4 rebounds on 61 percent shooting. He exploded in the last three games of the series, scoring 71 points on 24-for-35 shooting.

• Best Comeback Athlete: Russell Westbrook, who missed two months because of a reinjured knee, then returned to the Thunder lineup to help Oklahoma City finish the season with a 59-23 record and the second seed in the Western Conference.

• Best NHL Player: Sidney Crosby, who led the NHL with 104 points on the season, including a league-high 68 assists, 17 points ahead of his closest competitor.

• Best Driver: Ryan Hunter-Reay, who became the first American to win the Indianapolis 500 since 2006.

• Best International Athlete: Cristiano Ronaldo, the star forward for Portugal and Real Madrid who scored 66 goals in 56 games on his way to winning FIFA's Ballon d'Or Award as World Player of the Year.

• Best Male College Athlete: Doug McDermott, winner of the Wooden, U.S. Basketball Writers Association and AP Player of the Year awards who led the country in scoring, averaging 26.7 points per game on 52.6 percent shooting from the field. (Skip)

Fig. 4.16 Example 1. Pythia-1.4B data valuation result. LoGra captures the broad topic of soccer but lacks the specificity (except for the third most valuable data, which states that Christiano Ronaldo is the best soccer play who won the Ballon d'Or award).

Prompt

The healthcare system in the United States is

Response

a complex web of interrelated systems and organizations. The complexity has increased as more people have access to health care, but it also means that there are many different types or levels at which patients can be seen by providers (e...
Read More »http://www2a-1b3c4d5f9j0x6q8z7k/wpcontent/_media//img_1280×720sizesavetopbarlogoiconlarge

First Most Valuable Data

(Skip) Advertisement

Shelby has also had some stranger ventures, including a stint as a chicken farmer, founding a Texas chili festival, and once having a brand of deodorant named after him, Carroll Shelby's Pit-Stop.

In recent years, health has been an issue, with Shelby receiving a heart transplant in 1991. The experience of the transplant was so powerful to Shelby that he started the Carroll Shelby Children's Foundation to help children in need of organ transplants. (Skip)

Second Most Valuable Data

7866/USA-TODAY-Comics","image_url":"https://imgv2-2-f.scribdassets.com/img/document/313782330/original/49x65/f14f585214/1550554917?v=1","small_image_url":"https://imgv2-2-f.scribdassets.com/img/document/313782330/original/49x65/f14f585214/1550554917?v=1","medium_image_url":"https://imgv2-1-f.scribdassets.com/img/document/313782330/original/72x93/2b60ef2573/1550554917?v=1","large_image_url":"https://imgv2-1-f.scribdassets.com/img/document/313782330/original/114x151/32dc1e65cb/1550554917?v=1","large_embed_image_url":"https://imgv2-1-f.scribdassets.com/img/document/313782330/original/183x250/57d838b60c/1550554917?v=1","small_embed_image_url":"https://imgv2-1-f.scribdassets.com/img/document/313782330/120x164/55cdb12997/1550554917?v=1"}},{"document":{"id":291078998,"title":"The Dark Knight III Exclusive Preview","description":"","word_user_name":"USA TODAY Comics","word_user_url":"https://www.scribd.com/publisher/55187866/USA-TODAY-Comics","image_url":"https://imgv2-2-f.scribdassets.com/img/document/291078998/original/49x65/b049d57f76/1550578642?v=1","small_image_url":"https://imgv2-2-f.scribdassets.com/img/document/291078998/original/49x65/b049d57f76/1550578642?v=1","medium_image_url":"https://imgv

Third Most Valuable Data

image_url":"https://imgv2-2-f.scribdassets.com/img/document/270833503/original/49x65/8e38af2f71/1550578642?v=1","medium_image_url":"https://imgv2-2-f.scribdassets.com/img/document/270833503/original/72x93/6d93f91454/1550578642?v=1","large_image_url":"https://imgv2-2-f.scribdassets.com/img/document/270833503/original/114x151/519f278aff/1550578642?v=1","large_embed_image_url":"https://imgv2-2-f.scribdassets.com/img/document/270833503/original/183x250/1d9c4232c1/1550578642?v=1","small_embed_image_url":"https://imgv2-2-f.scribdassets.com/img/document/270833503/120x164/cd38f6cff3/1550578642?v=1"}},(Skip)

Fig. 4.17 Example 2. Pythia-1.4B data valuation result. We suspect that the random url in the model output dominates the query gradient and affects the data valuation result.

Prompt

Does artificial intelligence eliminate the need for human intelligence?

Response

The answer is no. Artificial Intelligence (AI) does not replace humans, but it can help us to do our jobs better and faster by automating some of them or even replacing a few tasks that are currently done manually in many organizations today:

First Most Valuable Data

The answer is “yes.”It is the only path forward for us as a species; we really do not have a choice. This then becomes a matter of when and how. When will we invent the technologies to not only stop the output of greenhouse gasses, but to reverse them? This is known as geoengineering, and it holds the true promise to fix the problems we have created. There are proposals to have autonomous ships sailing around the ocean that spray salt water, sulfur dioxide, or other particles that can reflect light back into the atmosphere or absorb some of it so the planet does not warm. There are other devices like filtrations systems that can pull carbon dioxide out of the air (like trees do), and sequester it, so that we can reduce the existing levels. A company called CarbonEngineering is already doing this and is expanding as we speak. I truly think that this is the future for us, a future where we spend our resources on fixing the planet and bringing it back to a pristine state. The jobs of the future will likely be related to restoring the planet, re-seeding the world’s biodiversity, restocking our oceans, and making our air breathable again. When people can stop having wars over resources, fighting for oil or water or food, we can bring humanity together to focus on the big picture: Our continued survival as a whole.(Skip)

Second Most Valuable Data

(Skip) Now, Artificial Intelligence (AI) has come to the rescue, in this, as in so many other computation-intensive tasks. An Indian startup, co-founded by three engineering dropouts, has harnessed AI, to power a tool, that will significantly reduce manual work involved in such tasks. It will come as a blessing for animation and game developers everywhere.

The tool — Norah.AI — is the work of Vrushali Prasade, Shubham Mishra and Harikrishna Valiyath (all three coincidentally aged 23, at the time), who quit BITS Pilani, Goa in 2015 and co-founded the startup, Absentia, in Bangalore. Norah is arguably, India’s first game workflow (the process of creating a game or an animation from story line to end product), driven by Neural Networks, Deep Learning and Artificial Intelligence. (Skip)

Fig. 4.18 Example 3. Pythia-1.4B data valuation result.

Prompt

What are the pros and cons of having robots in every home?

Response

The Pros: Robots can do a lot more than humans. They have better eyesight, they don't get tired easily or need to sleep much (they're not like us), their hands aren't as clumsy when it comes time for them take careof things around you etc… The Cons : You will be paying an arm & leg each month just so that your robot does what its programmed too! It's also very expensive if one is going buy all these machines from China which has been doing this since forever now.. I'm sure there must still some people who would rather pay $1000+ per year on top offrobot service but then again we live here... We'll see how long our government lasts with such policies!!

First Most Valuable Data

Former DeKalb County State's Attorney Clay Campbell attended Wednesday's sentencing and said it was a sad end to an event that demanded greater justice. Campbell, who became State's Attorney in December 2010 shortly after Curl was charged, said he was sorry he could not complete his commitment to the Keller family and noted their absence Wednesday was likely a sign of how they felt about the agreement.
From bone fragments that tested positive for Keller's DNA to the scratches on Curl's chest, his multiple accounts of events and his attempt to flee to Mexico and Louisiana, Campbell said the evidence and timeline would have held up at trial.
"It very likely would have been the longest trial in DeKalb County history," he said. "I was 100 percent confident in the evidence."
Campbell said it is just as important for the state's attorney to pursue justice to the maximum extent of the law for victims as it is to protect the public.
Keller's family and supporters were not the only ones displeased with the agreement.
Wednesday's sentencing started with a plea from Moria Curl to her brother as she shouted for him to turn down the deal and fight for his freedom.
"They're railroading you!" she shouted to her brother as she was escorted from the courtroom.
The reaction came as somewhat of a surprise to DeKalb County Public Defender Tom McCulloch, who said his client was at peace with his decision Wednesday morning. McCulloch said Curl maintains his innocence in an Alford plea while having a chance at life outside of prison with his scheduled release date to come when he is 71 years old.
In an Alford plea, the defendant maintains innocence but admits the evidence could convince a judge or a jury to find him guilty.

Second Most Valuable Data

(Skip) It's also been clear, he said, that law enforcement officers aren't fully aware of the requirements placed on them by state and federal gun laws. Harden's letter seemed like a final warning, Silvoso said.

"Here's our warning to you that we know you're doing it, and we want you to stop," he said. "And woe be unto you from this day forward if you are going to continue to do this after we've warned you."

After the ATF investigation in the Sacramento area, then-Assemblyman Roger Dickinson sponsored a bill in 2012 that would have allowed officers to buy off-roster guns but not resell them, closing an exception in the law.

The Sacramento Democrat said the law didn't make sense: guns which the state had decided were not safe could be legally purchased by law enforcement officers and then be sold to anyone — essentially putting banned guns into the state, undercutting the purpose of the safe gun list.

"It was hard to come up with a rational justification for it," Dickinson, who left the Assembly in 2014, said Friday. (Skip)

Fig. 4.19 Example 4. Pythia-1.4B data valuation result.

We observe that most valuable data identified by LoGra, especially for Llama3-8B-Instruct and GPT2-XL, share qualitative similarities (*e.g.*, semantics, style, token overlaps) with the queried LLM outputs. For instance, given Llama3's response on the dream manipulation product, LoGra identifies a scientific article that studies manually inducing the lucid dream as most valuable data in Figure 4.4a. In Figure 4.4b, both the GPT2-XL output and the corresponding most valuable data discuss the need for reducing emissions in the coal industry and its connection to the specific administration. In Figure 4.4c, the concept of "lifting barbell or dumbells" appear in the model output and the most valuable data.

However, we also notice several failure cases where the identified most valuable data seemingly do not share qualitative similarities with the LLM output, especially with Pythia-1.4B. We here provide three potential explanations on these failing examples based on our experiments. First, attributed data may lack qualitative similarities when the queried LLM output itself is incoherent that its gradient does not encode meaningful information. This aligns with our observation that the failure case occurs more frequently with lower-tier models like Pythia-1.4B whose outputs generally are of lower quality. Second, since we only used a 1B-token subset for data valuation, it is possible that our valuation dataset may lack similar data to some queries. As noted above, our experiment was largely limited by the storage of our cluster (not by the compute), so exploring data valuation on an industry-scale cluster would be interesting future work. Third, we posit that train/test gradients in influence functions may encode diverse information including features that are hardly perceptible to humans [78]. Therefore, it is possible that attributed data are indeed valuable for increasing the likelihood of the queried output by contributing to these other aspects while sharing little qualitative similarities.

To provide a slightly more formal argument on the third point in the above discussion, we note that influence functions tend to give a high score for the example that contributes most to decreasing (test) loss at the *current weight* [9]. At the same time, it is also hypothesized that different layers learn different concepts at different stages of training [21]. Combining these two facts, when interpreting influence analysis results, we need to think about which features the model is most likely learning at the current weight. Here, we specifically discuss two factors: training data quality and training steps. First, if the training data quality is low, then there would be a lot of features (*e.g.*, random email address) that are frequent enough in the training dataset to be considered as learnable patterns. In other words, even though these features look redundant to humans, they may still be useful for decreasing loss from the model perspective. Second, many LLMs are only pretrained for a single epoch, or under-trained to their pretraining dataset. That being said, redundant features from the first point would likely still remain as learning-worthy features at the end of training and are captured by influence functions. In sum, we hypothesize that as the model is well-trained on a high-quality dataset, influence functions would capture more similar data to the query LLM output. This hypothesis may also explain the observation from Grosse et al. [55] that most valuable data identified by influence functions on larger models tend to share more semantic similarity with results on smaller models, noting that larger models tend to converge faster to the point where they can only further decrease loss by learning high-level features. With this,

we present our experiment results with Pythia-1.4B below. (some of them are not totally bad, but mostly lack specificity to be considered as "most" valuable data to humans)

4.5 Related Work

Data Valuation. Measuring the value (or contribution) of training data on the model outputs has gained lots of attention recently. Exemplified by Data Shapley [51], a flurry of prior work [81, 92, 163] proposed exploiting the Shapley value or concepts from cooperative game theory to address the data valuation problem. However, most existing approaches in this line require repeated retraining of the model, a cost of which is hardly affordable even with small models. In addition to game-theoretic approaches, data valuation has also been tackled using reinforcement learning [175], meta learning [22], and training-free methods [114, 169]. Nevertheless, these works either suffer from high complexity from the need to train other models [22, 175] or high computational costs [114]. We direct readers to Sim et al. [146] for a more extensive survey on diverse data valuation approaches.

Influence Functions. Influence functions, a classic concept from robust statistics [63], estimate the infinitesimal effect of removing or adding a training data point without model retraining. They have various applications in machine learning, such as interpreting the model's behavior [64, 117, 55] and curating training datasets [102, 40]. However, when applied to large neural networks, the computation of the iHVP and its dot product with all training examples introduce scalability challenges. Besides gradient projection, past works have explored computing influence functions only on the last (few) layers [87, 142] to mitigate these challenges. However, subsequent works [41, 55] have shown that the influence on only a subset of layers is insufficient to capture the overall influence of a training data point. To avoid computing the gradient of all training examples, various filtering strategies, such as using the similarity in the model's representation space [60] or TF-IDF [174, 55], have also been proposed. While it is possible to adopt these filtering strategies for LoGra, they may introduce bias in the selection of the most influential sequences. For example, filtering candidate training sequences with TF-IDF might miss interesting influential sequences that do not share many tokens but are semantically related. Recently, similarly to LoGra, DataInf [91] and LESS [170] proposed using LoRA to efficiently compute influence functions. However, these approaches are only applicable in finetuning settings, whereas LoGra also supports influence analyses for pretraining.

4.6 Conclusion

In this work, we explored scaling data valuations with influence functions to billion-scale models and datasets as a potential technical solution to properly credit or compensate data providers for training LLMs. Toward this goal, we developed a novel gradient projection algorithm that can significantly improve the scalability of influence functions, and designed a simple and interoperable software. Our experiments showed that LoGra achieves competitive accuracy to other more expensive baselines on counterfactual evaluations, while

efficiently scaling to billion-scale models and datasets, thereby demonstrating the initial potential of the practical data valuation system.

Chapter 5

Conclusion

In summary, this thesis is motivated by the observation that the ML models are oftentimes reflections of their training data or (more broadly) inductive biases baked into optimization procedures. Building upon this belief, we attempted to operationalize scientific understanding of training data in large-scale ML by building gradient-based algorithmic frameworks while addressing scalability and implementation challenges stemming from the high-dimensional nature of gradients with advanced systems and ML tooling knowledge.

Specifically, in Chapter 2, we developed an automatic differentiation framework for multi-level optimization that abstracts away the complicated implementation of chain rules involving best-response Jacobian while providing diverse systems support for scalability. Notably, re-implementations of several popular MLO programs using our framework achieved up to 11% increase in test accuracy, 14% decrease in GPU memory usage, and 20% decrease in training wall time. Building on this framework, in Chapter 3, we developed a first-order implicit differentiation algorithm that is compatible with a wide range of adaptive optimizers and efficient distributed training techniques. When applied to various data optimization tasks, SAMA demonstrated up to 1.7/4.8× increase in throughput and 2.0/3.8× decrease in GPU memory usage respectively on single-/multi-GPU setups. Despite its efficiency, SAMA still achieved state-of-the-art results in both small- and large-scale data pruning experiments. Finally, in Chapter 4, we significantly improved memory/compute efficiency of influence functions by developing a novel gradient compression mechanism LoGra and designed interoperable software LogIX that can convert most existing training code into data attribution/valuation code. In our data influence analysis experiments, LoGra achieves competitive accuracy against more expensive baselines while showing up to 6,500× improvement in throughput and 5× reduction in GPU memory usage when applied to Llama3-8B-Instruct and the 1B-token dataset. We conclude this thesis by outlining promising research directions for future work that can be extended from this thesis below.

5.1 Data Curation for Foundation Models

In a near term, the most promising application of our research is to operationalize the data curation process for large-scale ML. As we enter the era of foundation models, the training

dataset size has grown massively to an extent that no humans are able to manually detect various pathologies in training data that could lead to undesired behaviors of the final ML model. For example, if the number of machine learning articles significantly outweighs the number of philosophy articles, training on this dataset could lead to overfitting where the model performs poorly when philosophy-related questions are asked. The challenge here is that one needs to understand the distribution of the entire dataset that may consist of trillions of tokens to detect such long-tail distribution issues as each of the training example may not look wrong *per se*. Accordingly, there is an increasing need for a systematic and principled approach to understand and optimize the effect of each data point on the final model at scale.

Technically, the data curation problem can be approached with both (generalized) meta learning in Chapter 2 & 3 and influence functions in Chapter 4. In fact, if we limit our focus to data pruning & selection, generalized meta learning is mathematically equivalent with iteratively applying influence functions in an online manner. However, each approach faces several technical challenges. First, generalized meta learning tends to suffer from instability due to the Hessian approximation. In our research in influence functions, we observe that the algorithmic instability involving the Hessian can be largely mitigated by using the (empirical) Fisher information matrix approximation. Thus, incorporating lessons learned in Chapter 4 back to generalized meta learning is one straightforward direction we may explore. Second, the programming interface for generalized meta learning may lack flexibility to support users' arbitrary ML codes. Due to its multi-level optimization formulation, generalized meta learning requires users to define multiple optimization problems in their code, whereas most ML practitioners implement a single optimization problem in their application. Designing an intuitive and interoperable programming interface as well as system infrastructures for generalized meta learning remains as an open problem. Lastly, practical data curation with influence functions requires further system research/optimization. In essence, influence functions translate data curation problems into vector similarity analysis problems, where each data (or group of data) is represented with its (average) gradient. Given the sheer amount of training data for foundation models, extensive system optimizations for fast vector similarity analysis are necessary.

5.2 Data Valuation & Attribution

In a longer term, scientific understanding of training data can be used to address diverse societal issues. One prominent example is the data compensation issue. As the capabilities of foundation models improve over time, it is natural to expect that human labors are going to be increasingly automated by these models, potentially disrupting the whole economy system. While the solution concept like universal basic income (UBI) where the government redistributes economic gains generated by these models to each citizen has been discussed, another potential solution is to redistribute economic gains to people based on the contributions of their data to the final model. In easier words, if someone provides highly valuable data in a large quantity, then they would be compensated more for this contribution. Notably, building a fair compensation system for the data provider or the *data market* may also solve the emerging problem of training the ML model on the copyrighted data.

From the technical viewpoint, building a fair data market requires accurately quantifying the contribution or *value* of each training data in the inference phase. While our work on data valuation with influence functions from Chapter 4 can be a natural solution to this problem, we also note that most inference stacks are highly optimized for forward-only computations whereas influence functions typically require backward computations. Therefore, a promising research direction is developing a variant of influence functions that avoid explicit backward computations. Another important future research direction for practical data valuation is to reduce the cost of influence function computations. To enable a fair compensation, we may need to compute influence scores for all outputs generated by the model. At the same time, several foundation models already handle millions or billions of queries on a daily basis. While our LOGRA significantly improves efficiency of influence computations via efficient gradient projection, it still is unable to handle even a single query batch in a real-time manner. We expect that the cost issue in influence functions can be approached from both algorithm and system perspectives. For algorithmic improvements, we consider more aggressive gradient compression (without major performance degradation) and amortization as two most promising directions. For system improvements, we are eager to explore various indexing techniques in (vector) database literature that can reduce time complexity of vector similarity search from $O(n)$ to $O\log(n)$ where n is the dataset size.

Appendix A

Appendix for Chapter 3

A.1 Code Example

Here, we provide simplified code for our experiments from Section 2.5. Note that every experiment shares a similar code structure when implemented with BETTY.

A.1.1 Data Reweighting for Class Imbalance

```
train_loader, valid_loader = setup_dataloader()
rwt_module, rwt_optimizer = setup_reweight()
cls_module, cls_optimizer, cls_scheduler = setup_classifier()

# Level 2
class Reweight(ImplicitProblem):
    def training_step(self, batch):
        inputs, labels = batch
        outputs = self.classifier(inputs)
        return F.cross_entropy(outputs, labels)

# Level 1
class Classifier(ImplicitProblem):
    def training_step(self, batch):
        inputs, labels = batch
        outputs = self.module(inputs)
        loss = F.cross_entropy(outputs, labels, reduction="none")
        loss_reshape = torch.reshape(loss, (-1, 1))
        # Reweighting
        weight = self.reweight(loss_reshape.detach())
        return torch.mean(weight * loss_reshape)
```

```
upper_config = Config(type="darts", retain_graph=True)
lower_config = Config(type="default", unroll_steps=5)

reweight = Reweight(name="reweight",
                    config=upper_config,
                    module=rwt_module,
                    optimizer=rwt_optimizer,
                    train_data_loader=valid_loader)
classifier = Classifier(name="classifier",
                        config=lower_config,
                        module=cls_module,
                        optimizer=cls_optimizer,
                        scheduler=cls_scheduler,
                        train_data_loader=train_loader)

probs = [reweight, classifier]
u2l = {reweight: [classifier]}
l2u = {classifier: [reweight]}
depends = {"l2u": l2u, "u2l": u2l}

engine = Engine(problems=probs, dependencies=depends)
engine.run()
```

Listing A.1 Simplified code of “Data Reweighting for Class Imbalance”

A.1.2 Correcting & Reweighting Corrupted Labels

```
train_loader, valid_loader = setup_dataloader()
rwt_module, rwt_optimizer = setup_reweight()
crt_module, crt_optimizer = setup_correct()
cls_module, cls_optimizer, cls_scheduler = setup_classifier()

# Level 2
class Correct(ImplicitProblem):
    def training_step(self, batch):
        inputs, labels = batch
        outputs, embeds = self.classifier(inputs, return_embeds=True)
        correct_outputs = self.module(embeds, test=True)
        ce_loss = F.cross_entropy(outputs, labels)
        aux_loss = F.cross_entropy(correct_outputs, labels)
        return ce_loss + aux_loss

# Level 2
class Reweight(ImplicitProblem):
    def training_step(self, batch):
        inputs, labels = batch
        outputs = self.classifier(inputs)
        return F.cross_entropy(outputs, labels)

# Level 1
class Classifier(ImplicitProblem):
    def training_step(self, batch):
        inputs, labels = batch
        outputs, embeds = self.module(inputs, return_embeds=True)
        # Correcting
        new_labels = self.correct(embeds, labels)
        log_softmax = F.log_softmax(outputs, dim=-1)
        loss = torch.sum(-log_softmax * new_labels, dim=-1)
        loss_reshape = torch.reshape(loss, (-1, 1))
        # Reweighting
        weight = self.reweight(loss_reshape.detach())
        return torch.mean(weight * loss_reshape)

upper_config = Config(type="darts", retain_graph=True)
lower_config = Config(type="default", unroll_steps=5)

correct = Correct(name="correct",
                  config=upper_config,
                  module=crt_module,
                  optimizer=crt_optimizer,
                  train_data_loader=valid_loader)
reweight = Reweight(name="reweight",
                    config=upper_config,
                    module=rwt_module,
                    optimizer=rwt_optimizer,
                    train_data_loader=valid_loader)
classifier = Classifier(name="classifier",
                        config=lower_config,
                        module=cls_module,
                        optimizer=cls_optimizer,
```

```
                        scheduler=cls_scheduler,
                        train_data_loader=train_loader)

probs = [correct, reweight, classifier]
u2l = {correct: [classifier], reweight: [classifier]}
l2u = {classifier: [correct, reweight]}
depends = {"l2u": l2u, "u2l": u2l}

engine = Engine(problems=probs, dependencies=depends)
engine.run()
```

Listing A.2 Simplified code of “Correcting & Reweighting Corrupted Labels”

A.1.3 Domain Adaptation for Pretraining & Finetuning

```
# Get module, optimizer, lr_scheduler, data loader for each problem
pt_module, pt_optimizer, pt_scheduler, pt_loader = setup_pretrain()
ft_module, ft_optimizer, ft_scheduler, ft_loader = setup_finetune()
rw_module, rw_optimizer, rw_scheduler, rw_loader = setup_reweight()

# Level 1
class Pretrain(ImplicitProblem):
    def training_step(self, batch):
        inputs, targets = batch
        outs = self.module(inputs)
        loss_raw = F.cross_entropy(outs, targets, reduction="none")

        logit = self.reweight(inputs)
        weight = torch.sigmoid(logit)
        return torch.mean(loss_raw * weight)

# Level 2
class Finetune(ImplicitProblem):
    def training_step(self, batch):
        inputs, targets = batch
        outs = self.module(inputs)
        loss = F.cross_entropy(outs, targets, reduction="none")
        loss = torch.mean(ce_loss)
        # Proximal regularization
        for (n1, p1), p2 in zip(self.module.named_parameters(), self.
            pretrain.module.parameters()):
            lam = 0 if "fc" in n1 else args.lam
            loss += lam * (p1 - p2).pow(2).sum()
        return loss

# Level 3
class Reweight(ImplicitProblem):
    def training_step(self, batch):
        inputs, targets = batch
        outs = self.finetune(inputs)
        return F.cross_entropy(outs, targets)

# Define optimization configurations
reweight_config = Config(type="darts", step=1, retain_graph=True)
finetune_config = Config(type="default", step=1)
pretrain_config = Config(type="default", step=1)
```

```
pretrain = Pretrain("pretrain", pt_config, pt_module, pt_optimizer
                    pt_scheduler, pt_loader)
finetune = Finetune("finetune", ft_config, ft_module, ft_optimizer
                    ft_scheduler, ft_loader)
reweight = Reweight("reweight", rw_config, rw_module, rw_optimizer
                    rw_scheduler, rw_loader)

probs = [reweight, finetune, pretrain]
u2l = {reweight: [pretrain]}
l2u = {pretrain: [finetune], finetune: [reweight]}
depends = {"u2l": u2l, "l2u": l2u}
engine = Engine(problems=probs, dependencies=depends)
engine.run()
```

Listing A.3 Simplified code of “Domain Adaptation for Pretraining & Finetuning”

A.1.4 Differentiable Neural Architecture Search

```
train_loader, valid_loader = setup_dataloader()
arch_module, arch_optimizer = setup_architecture()
cls_module, cls_optimizer, cls_scheduler = setup_classifier()

# Level 2
class Architecture(ImplicitProblem):
    def training_step(self, batch):
        x, target = batch
        alphas = self.module()
        return self.classifier.module.loss(x, alphas, target)

# Level 1
class Classifier(ImplicitProblem):
    def training_step(self, batch):
        x, target = batch
        alphas = self.architecture()
        return self.module.loss(x, alphas, target)

arch_config = Config(type="darts",
                     step=1,
                     retain_graph=True,
                     first_order=True)
cls_config = Config(type="default")

architecture = Architecture(name="architecture",
                            config=arch_config,
                            module=arch_module,
                            optimizer=arch_optimizer,
                            train_data_loader=valid_loader)
classifier = Classifier(name="classifier",
                        config=cls_config,
                        module=cls_module,
                        optimizer=cls_optimizer,
                        scheduler=cls_scheduler,
                        train_data_loader=train_loader)

probs = [architecture, classifier]
u2l = {architecture: [classifier]}
l2u = {classifier: [architecture]}
depends = {"l2u": l2u, "u2l": u2l}

engine = Engine(problems=probs, dependencies=depends)
```

```
engine.run()
```

Listing A.4 Simplified code of “Differentiable Neural Architecture Search”

A.2 Experiment Details

In this section, we provide further training details (*e.g.* hyperparameters) of each experiment.

A.2.1 Data Reweighting for Class Imbalance

Dataset We reuse the long-tailed CIFAR-10 dataset from the original paper [145] as our inner-level training dataset. More specifically, the imbalance factor is defined as the ratio between the number of training samples from the most common class and the most rare class. The number of training samples of other classes are defined by geometrically interpolating the number of training samples from the most common class and the most rare class. We randomly select 100 samples from the validation set to construct the upper-level (or meta) training dataset, and use the rest of it as the validation dataset, on which classification accuracy is reported in the main text.

Meta-Weight-Network We adopt a MLP with one hidden layer of 100 neurons (*i.e.* 1-100-1) as our Meta-Weight-Network (MWN). It is trained with the Adam optimizer [85] whose learning rate is set to 0.00001 throughout the whole training procedure, momentum values to (0.9, 0.999), and weight decay value to 0. MWN is trained for 10,000 iterations and learning rate is fixed throughout training.

Classification Network Following the original MWN work [145], we use ResNet32 [69] as our classification network. It is trained with the SGD optimizer whose initial learning rate is set to 0.1, momentum value to 0.9, and weight decay value to 0.0005. Training is performed for 10,000 iterations, and we decay the learning rate by a factor of 10 on the iterations of 5,000 and 7,500.

A.2.2 Correcting & Reweighting Corrupted Labels

Dataset We directly use TREC, AGNews, IMDB, SemEval, ChemProt, YouTube text classification datasets from the Wrench benchmark [177]. More specifically, we use the training split of each dataset for training the classification network, and the validation split for training the correcting and the reweighting networks. Test accuracy is measured on the test split.

Correct Network Our correct network takes the penultimate activation from the classification network, and outputs soft labels through the linear layer and the softmax layer. These

new soft labels are interpolated with the original labels via the reweighting scheme which is achieved with 2-layer MLP. As our reweighting network, the correct network is trained with Adam optimizer whose learning rate is set to 0.00001, momentum values to (0.9, 0.999), and weight decay value to 0.

Reweighting Network For our reweighting network, we reuse Meta-Weight-Net from the "Data Reweighting for Class Imbalance" experiment, follow all the training details.

Classification Network As our classification network, we adopt a 2-layer MLP with the hidden size of 100. The classification network is trained for 30,000 iterations with the SGD optimizer whose learning rate is set to 0.003, momentum to 0.9, and weight decay to 0.0001. Learning rate is decayed to 0 with the cosine annealing schedule during training.

A.2.3 Domain Adaptation for Pretraining & Finetuning

Dataset We split each domain of the OfficeHome dataset [159] into training/validation/test datasets with a ratio of 5:3:2. The pretraining network is trained on the training set of the source domain. Finetuning and reweighting networks are both trained on the training set of the target domain following the strategy proposed in [10]. The final performance is measured by the classification accuracy of the finetuning network on the test dataset of the target domain.

Pretraining Network We use ResNet18 [69] pretrained on the ImageNet dataset [33] for our pretraining network. Following the popular transfer learning strategy, we split the network into two parts, namely the feature (or convolutional layer) part and the classifier (or fully-connected layer) part, and each part is trained with different learning rates. Specifically, learning rates for the feature and the classifier parts are respectively set to 0.001 and 0.0001 with the Adam optimizer. They share the same weight decay value of 0.0005 and momentum values of (0.9, 0.999). Furthermore, we encourage the network weight to stay close to the pretrained weight by introducing the additional proximal regularization with the regularization value of 0.001. Training is performed for 1,000 iterations, and the learning rate is decayed by a factor of 10 on the iterations of 400 and 800.

Finetuning Network The same architecture and optimization configurations as the pretraining network are used for the finetuning network. The proximal regularization parameter, which encourages the finetuning network parameter to stay close to the pretraining network parameter, is set to 0.007.

Reweighting Network The same architecture and optimization configurations as the pre-training network are used for the reweighting network, except that no proximal regularization is applied to the reweighting network.

A.2.4 Differentiable Neural Architecture Search

Dataset Follwing the original paper [99], we use the first half of the CIFAR-10 training dataset as our inner-level training dataset (*i.e.* classification network) and the other half as the outer-level training dataset (*i.e.* architecture network). Training accuracy reported in the main text is measured on the CIFAR-10 validation dataset.

Architecture Network We adopt the same architecture search space as in the original paper [99] with 8 operations, and 7 nodes per convolutional cell. The architecture parameters are initialized to zero to ensure equal softmax values, and trained with the Adam optimizer [85] whose learning rate is fixed to 0.0003, momentum values to (0.5, 0.999), and weight decay value to 0.001 throughout training. Training is performed for 50 epochs.

Classification Network Given the above architecture parameters, we set our classification network to have 8 cells and the initial number of channels to be 16. The network is trained with the SGD optimizer whose initial learning rate is set to 0.025, momentum to 0.9, and weight decay value to 0.0003. Training is performed for 50 epochs, and the learning rate is decayed following the cosine annealing schedule without restart to the minimum learning rate of 0.001 by the end of training.

A.3 Systems Support

In this section, we perform additional analyses on the memory saving effects of our system features with two benchmarks: (1) differentiable neural architecture search and (2) data reweighting for class imbalance.

A.3.1 Differentiable Neural Architecture Search

	Baseline	+ mixed-precision
GPU Memory Usage	9867MiB	**5759MiB**

Table A.1 GPU memory usage analysis for DARTS.

A.3.2 Data Reweighting for Class Imbalance

In this experiment, we use ResNet50 [69] instead of ResNet30, to better study the memory reduction from our system features, when the larger model is used. Importantly, we also test the data-parallel training feature in addition to the mixed-precision training feature.

	Baseline	+ mixed-precision	+ data-parallel (2 GPUs)
GPU Memory Usage	6817MiB	4397MiB	**3185/3077MiB** (GPU0/1)

Table A.2 GPU memory usage analysis for MWN with ResNet-50.

As shown above, we observe more reduction in memory usage as we add more system features.

Appendix B

Appendix for Chapter 4

B.1 Philosophies behind SAMA & Scalable Meta Learning

Here, we additionally discuss several important design principles and philosophies behind scalable meta learning and SAMA in a Q&A format.

Q. Why do we study scalable meta learning?

A. Richard Sutton points out in his article "The Bitter Lesson" [153] that machine learning algorithms that stand the test of time are ones that continue to *scale gracefully with the increased computation budget* (*i.e.*, scalable algorithms). Given that meta learning is an important topic in machine learning with many applications including data optimization [136], hyperparameter optimization [46], few-shot learning [43, 130], and adversarial learning [111], it was a natural call for us to investigate the scalability of meta learning algorithms following the spirit of "The Bitter Lesson". Interestingly, such a focus on the scalability of meta learning algorithms distinguishes our work from most other meta learning works, in which the typical focus is to improve the overall performance of meta learning algorithms *under a limited computation budget* (usually bounded by a single GPU).

Q. What are the design principles behind scalable meta learning?

A. The increased computation budget powered by hardware advancements (*e.g.*, Moore's law) has evolved a new ecosystem of large models and datasets in machine learning over time, which involves both systems and algorithms components. For example, to efficiently leverage the increased computation for large-scale learning, diverse systems techniques, such as data/model/pipeline parallelism have been developed [75, 93, 144]. At the same time, researchers have devised various algorithms that are highly effective for large-scale learning, such as backpropagation [140], skip connections [69], Adam optimizer [85], self-attention [158], etc. Accordingly, in addition to guaranteeing memory/compute efficiency for

scalability, our major design principle for scalable meta learning was to *ensure compatibility with existing systems and algorithms* in the large-scale learning ecosystem.

Systems compatibility Given that a great deal of systems support in machine learning, such as communication-computation overlap [93], has been developed for first-order gradient methods, avoiding explicit computations of higher-order gradient information including Hessian-vector products was an important design principle in SAMA. Even though we mostly explored distributed training in this work, SAMA is also compatible with other system features such as half-precision training and activation checkpointing, which could further improve memory efficiency.

Algorithms compatibility While there exist several meta learning algorithms that avoid the computation of higher-order gradient information [90, 99, 97], many of these algorithms either assume the use of a naive SGD update rule or devise specific update rules tailored to their own algorithms at the base level, significantly hampering their algorithm compatibility. In contrast, SAMA allows for the use of arbitrary optimizers at the base level via algorithmic adaptation.

B.2 Experiment Details

In this section, we discuss various experiment details such as hyperparameters, baselines, and compute resources used for our experiments in Section 4. Our experiment codes are available in the supplementary material as well as under the official Betty GitHub repository [1].

B.2.1 Noisy Finetuning of Large Language Models

Hyperparameters We ran training for 1000 iterations on TREC/SemEval/IMDB/ChemProt/Yelp/ AGNews datasets from the WRENCH benchmark [177], with a batch size of 32, a weak supervision algorithm of majority voting, and the hyperparameters in Table B.1 below.

	model	optimizer	init_lr	lr_scheduler	wdecay	dataset	unroll step	SAMA α
Base	BERT-base	Adam	1e-5	cosine	0	WRENCH train set (with majority voting)	10	1.0
Meta (Reweight)	2-layer MLP	Adam	1e-5	None	0	WRENCH dev set	N/A	N/A
Meta (Correct)	2-layer MLP	Adam	1e-5	None	0	WRENCH dev set	N/A	N/A

Table B.1 Hyperparameters for *noisy finetuning of large language models* experiments.

Baselines We adopted naive finetuning and self-training (*i.e.*, COSINE [176]) approaches from the original WRENCH benchmark paper [177] as our baseline.

Compute Resources We used 1 NVIDIA RTX 2080Ti GPU for the main experiment, and 4 NVIDIA Tesla V100 GPUs for the throughput-memory analysis in Table 2 and Figure 1.

B.2.2 Continued Pretraining of Large Language Models

Hyperparameters We ran training for 100 epochs with a batch size of 16, a maximum sequence length of 256, and the hyperparameters in Table B.2 below.

Baselines We adopt DAPT [62] and TARTAN-MT [34] as our baselines for this experiment. In detail, DAPT [62] performs additional masked language model pretraining on domain-specific data on top of the pretrained RoBERTa-base model and then finetunes the model on the downstream text classification task. We follow [62] (see Table 14 in the original paper) for setting downstream finetuning hyperparameters. Alternatively, TARTAN-MT [34] performs masked language modeling with task specific data and downstream text classification training simultaneously in a multitask fashion through two different heads.

[1] https://github.com/leopard-ai/betty/tree/main/examples

	model	optimizer	init_lr	lr_scheduler	wdecay	dataset	unroll step	SAMA α
Base (Downstream)	RoBERTa-base	Adam	2e-5	linear decay + warmup linear (warmup proportion 0.6)	0	train split of ChemProt/HyperPartisan/ ACL-ARC/SciERC	10	0.3
Base (Auxiliary)	RoBERTa-base	Adam	2e-5	linear decay + warmup linear (warmup proportion 0.6)	0	train split of ChemProt/HyperPartisan/ ACL-ARC/SciERC	10	0.3
Meta	2-layer MLP	Adam	1e-5	None	0	train split of ChemProt/HyperPartisan/ ACL-ARC/SciERC	N/A	N/A

Table B.2 Hyperparameters for *continued pretraining of large language models* experiments.

Compute Resources We used 1 NVIDIA Tesla V100 GPU for the main experiment, and 1 NVIDIA RTX A6000 GPU for the "memory vs model-size analysis" in Figure 1.

B.2.3 Scale-Agnostic Efficient Data Pruning

Hyperparameters We ran meta learning for 30 epochs with a batch size of 256 and the configuration shown in Table B.3 below. After pruning data based on the meta learning result, we ran ImageNet-1k training for 120 epochs with the learning rate decayed by 10 at epochs [40, 80] following the set up in DynaMS [162].

	model	optimizer	init_lr	lr_scheduler	wdecay	dataset	unroll step	SAMA α
Base	ResNet-50	SGD	1e-1	None	1e-4	ImageNet-1k train set	2	1.0
Meta	2-layer MLP	Adam	1e-5	None	0	ImageNet-1k train set	N/A	N/A

Table B.3 Hyperparameters for *ImageNet-1k data pruning* experiments

For the CIFAR-10 data pruning experiment, we ran meta learning for 50 epochs with a batch size of 128, and configuration in Table B.4 below. After pruning the data based on the meta learning result, we ran CIFAR-10 training for 200 epochs with the cosine learning rate decay schedule following the setup in DeepCore [59].

	model	optimizer	init_lr	lr_scheduler	wdecay	dataset	unroll step	SAMA α
Base	ResNet-18	SGD	1e-1	None	5e-4	CIFAR-10 train set	2	1.0
Meta	2-layer MLP	Adam	1e-5	None	0	CIFAR-10 train set	N/A	N/A

Table B.4 Hyperparameters for *CIFAR-10 data pruning* experiments

Baselines We adopt EL2N [119], GraNd [119], DynaMS [162] as our baselines for the ImageNet-1k experiments and GraNd [119], forgetting [155], margin [26] for the CIFAR-10 experiments. In detail, EL2N/GraND [119] respectively select samples with large L2-loss/gradient-norm values, forgetting [119] chooses samples that are frequently forgotten

during training, and margin [26] chooses samples with least confidence. While these baselines are considered *static pruning*, DynaMS [162] falls under the category of dynamic pruning where data to be pruned change during training. Dynamic pruning may see the whole training data across different epochs, making a fair comparison difficult. Surprisingly, despite being a static pruning algorithm, SAMA-based data pruning still achieves a better performance than DynaMS.

Compute Resources We used 4 NVIDIA Tesla V100 GPUs for Imagenet-1k data pruning meta learning experiments and 1 NVIDIA RTX 2080Ti GPU for CIFAR-10 experiments.

Additional Information We measured the uncertainty $\mathscr{U}$ via the difference between the predictions of the current model and the exponentially-moving-averaged model.

Appendix C

Appendix for Chapter 5

C.1 Code Examples

We provide a simplified code for our language modeling experiment from Section 4.4.2 to demonstrate usability of LoGra. LoGra is open-sourced under Apache 2.0 license here.

C.1.1 Log Extraction

```
import logix
from logix.statistic import Covariance

model, tokenizer, train_loader = setup()

# Initialize LogIX
run = logix.init(project="llm", config="config.yaml")

# Register the model
run.watch(model, type_filter=[nn.Linear], name_filter=["mlp"])

# Add LoGra
run.add_lora()

# Setup logging
run.setup("log": "grad", "save": "grad", "statistic": {"grad":
    Covariance})

# Start logging
for batch in train_loader:
    data_id = tokenizer.batch_decode(batch["input_ids"])
```

```
    targets = batch.pop("labels")
    with run(data_id=data_id, mask=batch["attention_mask"]):
        # User's existing training code
        model.zero_grad()
        lm_logits = model(**batch)
        shift_logits = lm_logits[..., :-1, :].contiguous()
        shift_labels = targets[..., 1:].contiguous()
        loss = F.cross_entropy(shift_logits.view(-1, shift_logits.
            size(-1)),
                               shift_labels.view(-1),
                               reduction="sum",
                               ignore_index=-100)
        loss.backward()

# Finalize logging
logix.finalize()
```

C.1.2 Influence Computation

```
import logix

model, tokenizer, test_loader = setup()

run = logix.init(project="llm", config="config.yaml")
run.watch(model, type_filter=[nn.Linear], name_filter=["mlp"])

# Load saved logs (e.g. train gradient & Hessian)
logix.initialize_from_log()
log_loader = logix.build_log_dataloader(batch_size=64)

logix.setup({"log": "grad"})
for batch in test_loader:
    data_id = tokenizer.batch_decode(batch["input_ids"])
    targets = batch.pop("labels")
    with run(data_id=data_id, mask=batch["attention_mask"]):
        model.zero_grad()
        lm_logits = model(**batch)
        shift_logits = lm_logits[..., :-1, :].contiguous()
        shift_labels = targets[..., 1:].contiguous()
        loss = F.cross_entropy(shift_logits.view(-1, shift_logits.
            size(-1)),
                               shift_labels.view(-1),
                               reduction="sum",
```

```
                                ignore_index=-100)
        loss.backward()

    # Get the (gradient) log for the current test batch
    test_log = run.get_log()

    # Compute influence scores (with l-RealtIF)
    influence_scores = run.compute_influence_all(test_log, log_loader
        , mode="cosine")
```

C.2 Experiment Details

For EKFAC influence [55] and LoGra, we set the damping term in influence functions as $0.1 \times$mean(eigenvalues) for all layers following the practice in Grosse et al. [55].

C.2.1 Quantitative Counterfactual Experiments

For all our quantitative counterfactual experiments, we project gradients onto a low-dimensional space using LoGra with $k_i = k_o = 128$. We used the same experimental setup, including the configurations for the baseline data valuation techniques, from Park et al. [117] and Bae et al. [8]. We used one A100 GPU with 80GB VRAM for all our counterfactual evaluation experiments. For model training, we used hyperparameters in Table C.1 for each experiment.

	FMNIST	**CIFAR-10**	**WikiText**
Model	3-layer MLP	ResNet-9	GPT2
Optimizer	SGD-M	SGD-M	AdamW
LR Scheduler	None	Cyclic	None
Learning Rate	3e-2	4e-1	3e-5
Weight Decay	1e-3	1e-3	1e-2
Batch Size	64	512	8
Sequence Length	N/A	N/A	512
Epochs	20	25	3

Table C.1 Hyperparameter used in experiments in Section 4.4

Brittleness Test. For classification tasks, we first selected 100 correctly classified test examples when the model is trained on the full dataset (across all 5 random seeds). Then, for each test example x_{te}, we identified the top-k influential data points using the data valuation algorithm, removed these training data points, retrained the model, and examined if this removal causes misclassification of x_{te} on average (across 3 random seeds). In Figure 4.3, we reported the fraction of test examples (out of 100) that get misclassified after removing at most k training data points. For the language modeling task, we selected the 50 test sequences, obtained the top influential training sequences using the data valuation method, and reported the mean test perplexity after removing the top-k influential sequences and retraining the model.

Linear Datamodeling Score (LDS). We measured LDS by generating 100 data subsets of size $|S_i| = |D|/2$. For each data subset, we retrained the model 10 times for FashionMNIST, 20 times for CIFAR-10, and 5 times for WikiText to construct the ground truth. The LDS results in Figure 4.3 show the mean and standard deviation of LDS obtained from 5 distinctly trained models. A more detailed description of the LDS evaluation can be found in Park et al. [117].

References

[1] Abbas, A. K. M., Tirumala, K., Simig, D., Ganguli, S., and Morcos, A. S. (2023). Semdedup: Data-efficient learning at web-scale through semantic deduplication. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

[2] AI@Meta (2024). Llama 3 model card.

[3] Allal, L. B., Li, R., Kocetkov, D., Mou, C., Akiki, C., Ferrandis, C. M., Muennighoff, N., Mishra, M., Gu, A., Dey, M., et al. (2023). Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

[4] Antoniou, A., Edwards, H., and Storkey, A. (2019). How to train your MAML. In *International Conference on Learning Representations*.

[5] Arnold, S. M., Mahajan, P., Datta, D., Bunner, I., and Zarkias, K. S. (2020). learn2learn: A library for meta-learning research. *arXiv preprint arXiv:2008.12284*.

[6] Axelsson, O. (1996). *Iterative solution methods*. Cambridge university press.

[7] Azerbayev, Z., Schoelkopf, H., Paster, K., Santos, M. D., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. (2023). Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

[8] Bae, J., Lin, W., Lorraine, J., and Grosse, R. (2024). Training data attribution via approximate unrolled differentiation.

[9] Bae, J., Ng, N., Lo, A., Ghassemi, M., and Grosse, R. B. (2022). If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967.

[10] Bai, Y., Chen, M., Zhou, P., Zhao, T., Lee, J., Kakade, S., Wang, H., and Xiong, C. (2021). How important is the train-validation split in meta-learning? In *International Conference on Machine Learning*, pages 543–553. PMLR.

[11] Baik, J. S. (2020). Data privacy against innovation or against discrimination?: The case of the california consumer privacy act (ccpa). *Telematics and Informatics*, 52.

[12] Barshan, E., Brunet, M.-E., and Dziugaite, G. K. (2020). Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pages 1899–1909. PMLR.

[13] Basu, S., Pope, P., and Feizi, S. (2020). Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*.

[14] Bengio, Y., Hinton, G., Yao, A., Song, D., Abbeel, P., Darrell, T., Harari, Y. N., Zhang, Y.-Q., Xue, L., Shalev-Shwartz, S., et al. (2024). Managing extreme ai risks amid rapid progress. *Science*, 384(6698):842–845.

[15] Bergamin, F., Mattei, P.-A., Havtorn, J. D., Senetaire, H., Schmutz, H., Maaløe, L., Hauberg, S., and Frellsen, J. (2022). Model-agnostic out-of-distribution detection using combined statistical tests. In *International Conference on Artificial Intelligence and Statistics*, pages 10753–10776. PMLR.

[16] Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. (2023). Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

[17] Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. (2021). Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*.

[18] Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. (2022). Efficient and modular implicit differentiation. *Advances in neural information processing systems*, 35:5230–5242.

[19] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[20] Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. (2022). Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*.

[21] Chen, Y., Yuille, A., and Zhou, Z. (2023). Which layer is learning faster? a systematic exploration of layer-wise convergence rate for deep neural networks. In *The Eleventh International Conference on Learning Representations*.

[22] Choe, S., Mehta, S. V., Ahn, H., Neiswanger, W., Xie, P., Strubell, E., and Xing, E. (2024a). Making scalable meta learning practical. *Advances in neural information processing systems*, 36.

[23] Choe, S. K., Ahn, H., Bae, J., Zhao, K., Kang, M., Chung, Y., Pratapa, A., Neiswanger, W., Strubell, E., Mitamura, T., et al. (2024b). What is your data worth to gpt? llm-scale data valuation with influence functions. *arXiv preprint arXiv:2405.13954*.

[24] Choe, S. K., Neiswanger, W., Xie, P., and Xing, E. (2023). Betty: An automatic differentiation library for multilevel optimization. In *The Eleventh International Conference on Learning Representations*.

[25] Clarke, R. M., Oldewage, E. T., and Hernández-Lobato, J. M. (2022). Scalable one-pass optimisation of high-dimensional weight-update hyperparameters by implicit differentiation. In *International Conference on Learning Representations*.

[26] Coleman, C., Yeh, C., Mussmann, S., Mirzasoleiman, B., Bailis, P., Liang, P., Leskovec, J., and Zaharia, M. (2020). Selection via proxy: Efficient data selection for deep learning. In *International Conference on Learning Representations*.

[27] Cook, R. D. (1977). Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18.

[28] Cook, R. D. (1986). Assessment of local influence. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 48(2):133–155.

[29] Cook, R. D. and Weisberg, S. (1980). Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508.

[30] Couellan, N. and Wang, W. (2016). On the convergence of stochastic bi-level gradient methods. *Optimization*.

[31] Cui, H. and Bai, J. (2019). A new hyperparameters optimization method for convolutional neural networks. *Pattern Recognition Letters*, 125:828–834.

[32] Deleu, T., Würfl, T., Samiei, M., Cohen, J. P., and Bengio, Y. (2019). Torchmeta: A Meta-Learning library for PyTorch. Available at: https://github.com/tristandeleu/pytorch-meta.

[33] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[34] Dery, L. M., Michel, P., Talwalkar, A., and Neubig, G. (2022). Should we be pre-training? an argument for end-task aware training as an alternative. In *International Conference on Learning Representations*.

[35] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[36] Dhillon, G. S., Chaudhari, P., Ravichandran, A., and Soatto, S. (2020). A baseline for few-shot image classification. In *International Conference on Learning Representations*.

[37] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

[38] Dubey, P., Neyman, A., and Weber, R. J. (1981). Value theory without efficiency. *Mathematics of Operations Research*, 6(1):122–128.

[39] Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., et al. (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12.

[40] Engstrom, L., Feldmann, A., and Madry, A. (2024). Dsdm: Model-aware dataset selection with datamodels. *arXiv preprint arXiv:2401.12926*.

[41] Feldman, V. and Zhang, C. (2020). What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891.

[42] Fernandez, R. C. (2023). Data-sharing markets: Model, protocol, and algorithms to incentivize the formation of data-sharing consortia. In *Proceedings ACMSIGMOD International Conference on Management of Data*.

[43] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

[44] Flener, P. and Schmid, U. (2008). An introduction to inductive programming. *Artificial Intelligence Review*, 29:45–62.

[45] Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pages 1165–1173. PMLR.

[46] Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pages 1568–1577. PMLR.

[47] Fung, S. W., Heaton, H., Li, Q., McKenzie, D., Osher, S., and Yin, W. (2022). Jfb: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6648–6656.

[48] Gadre, S. Y., Ilharco, G., Fang, A., Hayase, J., Smyrnis, G., Nguyen, T., Marten, R., Wortsman, M., Ghosh, D., Zhang, J., et al. (2023). Datacomp: In search of the next generation of multimodal datasets. *arXiv preprint arXiv:2304.14108*.

[49] Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

[50] Garg, B., Zhang, L., Sridhara, P., Hosseini, R., Xing, E., and Xie, P. (2022). Learning from mistakes–a framework for neural architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*.

[51] Ghorbani, A. and Zou, J. (2019). Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pages 2242–2251. PMLR.

[52] Gokaslan, A., Cohen, V., Pavlick, E., and Tellex, S. (2019). Openwebtext corpus.

[53] Grazzi, R., Franceschi, L., Pontil, M., and Salzo, S. (2020). On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pages 3748–3758. PMLR.

[54] Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. (2019). Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*.

[55] Grosse, R., Bae, J., Anil, C., Elhage, N., Tamkin, A., Tajdini, A., Steiner, B., Li, D., Durmus, E., Perez, E., Hubinger, E., Lukošiūtė, K., Nguyen, K., Joseph, N., McCandlish, S., Kaplan, J., and Bowman, S. R. (2023). Studying large language model generalization with influence functions.

[56] Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582. PMLR.

[57] Grynbaum, M. and Mac, R. (2023). The times sues openai and microsoft over a.i. use of copyrighted work. *New York Times*.

[58] Gudovskiy, D., Rigazio, L., Ishizaka, S., Kozuka, K., and Tsukizawa, S. (2021). Autodo: Robust autoaugment for biased data with label noise via scalable probabilistic implicit differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16601–16610.

[59] Guo, C., Zhao, B., and Bai, Y. (2022). Deepcore: A comprehensive library for coreset selection in deep learning. In *Database and Expert Systems Applications: 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part I*, pages 181–195. Springer.

[60] Guo, H., Rajani, N. F., Hase, P., Bansal, M., and Xiong, C. (2020). Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*.

[61] Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31.

[62] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of ACL*.

[63] Hampel, F. R. (1974). The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393.

[64] Han, X., Wallace, B. C., and Tsvetkov, Y. (2020). Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*.

[65] Hanawa, K., Yokoi, S., Hara, S., and Inui, K. (2020). Evaluation of similarity-based explanations. *arXiv preprint arXiv:2006.04528*.

[66] Hataya, R. and Yamada, M. (2023). Nyström method for accurate and scalable implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 4643–4654. PMLR.

[67] Hataya, R., Zdenek, J., Yoshizoe, K., and Nakayama, H. (2022). Meta approach to data augmentation optimization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2574–2583.

[68] He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738.

[69] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[70] He, X., Cai, Z., Wei, W., Zhang, Y., Mou, L., Xing, E., and Xie, P. (2021). Towards visual question answering on pathology images. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 708–718.

[71] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

[72] Hong, M., Wai, H.-T., Wang, Z., and Yang, Z. (2020). A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic. *arXiv preprint arXiv:2007.05170*.

[73] Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2021). Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

[74] Huang, J. and Chang, K. C.-C. (2023). Citation: A key to building responsible and accountable large language models. *arXiv preprint arXiv:2307.02185*.

[75] Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32.

[76] Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. (2019). Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*.

[77] Ilyas, A., Park, S. M., Engstrom, L., Leclerc, G., and Madry, A. (2022). Datamodels: Predicting predictions from training data.

[78] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32.

[79] Jaakkola, T. and Haussler, D. (1998). Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, 11.

[80] Ji, K., Yang, J., and Liang, Y. (2021). Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892. PMLR.

[81] Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gürel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. J. (2019). Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR.

[82] J.L. et al. v. Alphabet Inc. (2023). Case 3:23-cv-03416, N.D. Cal.

[83] Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

[84] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[85] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[86] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., et al. (2023). Segment anything. *arXiv preprint arXiv:2304.02643*.

[87] Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

[88] Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S., et al. (2020). Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*.

[89] Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

[90] Kwon, J., Kwon, D., Wright, S., and Nowak, R. D. (2023). A fully first-order method for stochastic bilevel optimization. In *International Conference on Machine Learning*, pages 18083–18113. PMLR.

[91] Kwon, Y., Wu, E., Wu, K., and Zou, J. (2024). Datainf: Efficiently estimating data influence in loRA-tuned LLMs and diffusion models. In *The Twelfth International Conference on Learning Representations*.

[92] Kwon, Y. and Zou, J. (2021). Beta shapley: a unified and noise-reduced data valuation framework for machine learning. *arXiv preprint arXiv:2110.14049*.

[93] Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., and Chintala, S. (2020). Pytorch distributed: experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13:3005–3018.

[94] Li, Z., Malladi, S., and Arora, S. (2021). On the validity of modeling SGD with stochastic differential equations (SDEs). In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

[95] Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., and Zemel, R. (2018). Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091. PMLR.

[96] Ling, R. F. (1984). Residuals and influence in regression.

[97] Liu, B., Ye, M., Wright, S., Stone, P., and qiang liu (2022). BOME! bilevel optimization made easy: A simple first-order approach. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[98] Liu, G.-H. and Theodorou, E. A. (2019). Deep learning theory review: An optimal control and dynamical systems perspective. *arXiv preprint arXiv:1908.10920*.

[99] Liu, H., Simonyan, K., and Yang, Y. (2019a). DARTS: differentiable architecture search. In *ICLR*.

[100] Liu, R., Liu, Y., Zeng, S., and Zhang, J. (2021a). Towards gradient-based bilevel optimization with non-convex followers and beyond. *Advances in Neural Information Processing Systems*, 34.

[101] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[102] Liu, Z., Ding, H., Zhong, H., Li, W., Dai, J., and He, C. (2021b). Influence selection for active learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9274–9283.

[103] Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR.

[104] Luketina, J., Berglund, M., Greff, K., and Raiko, T. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International conference on machine learning*, pages 2952–2960. PMLR.

[105] Maclaurin, D., Duvenaud, D., and Adams, R. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International conference on machine learning*, pages 2113–2122. PMLR.

[106] Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR.

[107] Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

[108] Metz, C. (2022). Lawsuit takes aim at the way A.I. is built. *New York Times*.

[109] Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., et al. (2022). Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*.

[110] Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. (2019). Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. PMLR.

[111] Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. (2017). Unrolled generative adversarial networks. In *International Conference on Learning Representations*.

[112] Migdalas, A., Pardalos, P. M., and Värbrand, P. (1998). *Multilevel optimization: algorithms and applications*, volume 20. Springer Science & Business Media.

[113] Nanda, N. and Bloom, J. (2022). Transformerlens. https://github.com/TransformerLensOrg/TransformerLens.

[114] Nohyun, K., Choi, H., and Chung, H. W. (2023). Data valuation without training of a model. In *The Eleventh International Conference on Learning Representations*.

[115] Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. (2022). In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

[116] Papernot, N., McDaniel, P., Sinha, A., and Wellman, M. P. (2018). Sok: Security and privacy in machine learning. In *2018 IEEE European symposium on security and privacy (EuroS&P)*, pages 399–414. IEEE.

[117] Park, S. M., Georgiev, K., Ilyas, A., Leclerc, G., and Madry, A. (2023). TRAK: Attributing model behavior at scale. In *International Conference on Machine Learning*, pages 27074–27113. PMLR.

[118] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[119] Paul, M., Ganguli, S., and Dziugaite, G. K. (2021). Deep learning on a data diet: Finding important examples early in training. *Advances in Neural Information Processing Systems*, 34:20596–20607.

[120] Pearlmutter, B. A. (1994). Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160.

[121] Pearlmutter, B. A. and Siskind, J. M. (2008). Reverse-mode ad in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.*, 30:7:1–7:36.

[122] Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International conference on machine learning*, pages 737–746. PMLR.

[123] Perez, E., Huang, S., Song, F., Cai, T., Ring, R., Aslanides, J., Glaese, A., McAleese, N., and Irving, G. (2022). Red teaming language models with language models. *arXiv preprint arXiv:2202.03286*.

[124] Pruthi, G., Liu, F., Kale, S., and Sundararajan, M. (2020). Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

[125] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2023). Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR.

[126] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[127] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

[128] Raghu, A., Lorraine, J., Kornblith, S., McDermott, M., and Duvenaud, D. K. (2021). Meta-learning to improve pre-training. *Advances in Neural Information Processing Systems*, 34.

[129] Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2020). Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

[130] Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32.

[131] Rajeswaran, A., Mordatch, I., and Kumar, V. (2020). A game theoretic framework for model based reinforcement learning. In *International conference on machine learning*, pages 7953–7963. PMLR.

[132] Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. (2019). Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR.

[133] Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. (2020). Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

[134] Ratner, A. J., De Sa, C. M., Wu, S., Selsam, D., and Ré, C. (2016). Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29.

[135] Razeghi, Y., RobertL.Logan, I., Gardner, M., and Singh, S. (2022). Impact of pre-training term frequencies on few-shot numerical reasoning. In *Conference on Empirical Methods in Natural Language Processing*.

[136] Ren, M., Zeng, W., Yang, B., and Urtasun, R. (2018). Learning to reweight examples for robust deep learning. In *International conference on machine learning*, pages 4334–4343. PMLR.

[137] Ren, Z., Yeh, R., and Schwing, A. (2020). Not all unlabeled data are equal: Learning to weight data in semi-supervised learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21786–21797. Curran Associates, Inc.

[138] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

[139] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. (2023). Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

[140] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[141] Sato, R., Tanaka, M., and Takeda, A. (2021). A gradient method for multilevel optimization. *Advances in Neural Information Processing Systems*, 34.

[142] Schioppa, A., Zablotskaia, P., Vilar, D., and Sokolov, A. (2022). Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8179–8186.

[143] Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C. W., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., Schramowski, P., Kundurthy, S. R., Crowson, K., Schmidt, L., Kaczmarczyk, R., and Jitsev, J. (2022). LAION-5b: An open large-scale dataset for training next generation image-text models. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[144] Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

[145] Shu, J., Xie, Q., Yi, L., Zhao, Q., Zhou, S., Xu, Z., and Meng, D. (2019). Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930.

[146] Sim, R. H. L., Xu, X., and Low, B. K. H. (2022). Data valuation in machine learning:" ingredients", strategies, and open challenges. In *IJCAI*, pages 5607–5614.

[147] Soldaini, L., Kinney, R., Bhagia, A., Schwenk, D., Atkinson, D., Authur, R., Bogin, B., Chandu, K., Dumas, J., Elazar, Y., et al. (2024). Dolma: An open corpus of three trillion tokens for language model pretraining research. *arXiv preprint arXiv:2402.00159*.

[148] Somayajula, S. A., Song, L., and Xie, P. (2022). A multi-level optimization framework for end-to-end text augmentation. *Transactions of the Association for Computational Linguistics*, 10:343–358.

[149] Song, H., Kim, M., Park, D., Shin, Y., and Lee, J.-G. (2022). Learning from noisy labels with deep neural networks: A survey. *IEEE transactions on neural networks and learning systems*, 34(11):8135–8153.

[150] Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., and Morcos, A. (2022). Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536.

[151] Stephan, M., Hoffman, M. D., Blei, D. M., et al. (2017). Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35.

[152] Such, F. P., Rawal, A., Lehman, J., Stanley, K., and Clune, J. (2020). Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *International Conference on Machine Learning*, pages 9206–9216. PMLR.

[153] Sutton, R. (2019). The bitter lesson. *Incomplete Ideas (blog)*, 13(1).

[154] Tang, K., Huang, J., and Zhang, H. (2020). Long-tailed classification by keeping the good and removing the bad momentum causal effect. *Advances in Neural Information Processing Systems*, 33:1513–1524.

[155] Toneva, M., Sordoni, A., des Combes, R. T., Trischler, A., Bengio, Y., and Gordon, G. J. (2019). An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*.

[156] Torralba, A. and Efros, A. A. (2011). Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE.

[157] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[158] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[159] Venkateswara, H., Eusebio, J., Chakraborty, S., and Panchanathan, S. (2017). Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5018–5027.

[160] Vicente, L. N. and Calamai, P. H. (1994). Bilevel and multilevel programming: A bibliography review. *Journal of Global optimization*, 5(3):291–306.

[161] Voigt, P. and Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555.

[162] Wang, J., Li, Y., Zhuo, J., Shi, X., ZHANG, W., Gong, L., Tao, T., Liu, P., Bao, Y., and Yan, W. (2023). DynaMS: Dyanmic margin selection for efficient deep learning. In *The Eleventh International Conference on Learning Representations*.

[163] Wang, J. T. and Jia, R. (2023). Data banzhaf: A robust data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 6388–6421. PMLR.

[164] Wang, Y., Guo, J., Song, S., and Huang, G. (2020). Meta-semi: A meta-learning approach for semi-supervised learning. *CoRR*, abs/2007.02394.

[165] Wang, Z., Dai, Z., Póczos, B., and Carbonell, J. (2019). Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302.

[166] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

[167] Worledge, T., Shen, J. H., Meister, N., Winston, C., and Guestrin, C. (2023). Unifying corroborative and contributive attributions in large language models. *arXiv preprint arXiv:2311.12233*.

[168] Wu, Z., Geiger, A., Arora, A., Huang, J., Wang, Z., Goodman, N. D., Manning, C. D., and Potts, C. (2024). pyvene: A library for understanding and improving pytorch models via interventions. *arXiv preprint arXiv:2403.07809*.

[169] Wu, Z., Shu, Y., and Low, B. K. H. (2022). Davinz: Data valuation using deep neural networks at initialization. In *International Conference on Machine Learning*, pages 24150–24176. PMLR.

[170] Xia, M., Malladi, S., Gururangan, S., Arora, S., and Chen, D. (2024). Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*.

[171] Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. (2015). Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2691–2699.

[172] Xie, P. and Du, X. (2022). Performance-aware mutual knowledge distillation for improving neural architecture search. *CVPR*.

[173] Xie, S. M., Santurkar, S., Ma, T., and Liang, P. (2023). Data selection for language models via importance resampling. *arXiv preprint arXiv:2302.03169*.

[174] Yeh, C.-K., Taly, A., Sundararajan, M., Liu, F., and Ravikumar, P. (2022). First is better than last for language data influence. *Advances in Neural Information Processing Systems*, 35:32285–32298.

[175] Yoon, J., Arik, S., and Pfister, T. (2020). Data valuation using reinforcement learning. In *International Conference on Machine Learning*, pages 10842–10851. PMLR.

[176] Yu, Y., Zuo, S., Jiang, H., Ren, W., Zhao, T., and Zhang, C. (2021). Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1063–1077.

[177] Zhang, J., Yu, Y., Li, Y., Wang, Y., Yang, Y., Yang, M., and Ratner, A. (2021a). Wrench: A comprehensive benchmark for weak supervision. *arXiv preprint arXiv:2109.11377*.

[178] Zhang, M., Su, S. W., Pan, S., Chang, X., Abbasnejad, E. M., and Haffari, R. (2021b). idarts: Differentiable architecture search with stochastic implicit gradients. In *International Conference on Machine Learning*, pages 12557–12566. PMLR.

[179] Zhao, B., Lyu, B., Fernandez, R. C., and Kolar, M. (2023). Addressing budget allocation and revenue allocation in data market environments using an adaptive sampling algorithm. In *International Conference on Machine Learning*, pages 42081–42097. PMLR.

[180] Zheng, G., Awadallah, A. H., and Dumais, S. (2021). Meta label correction for noisy label learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11053–11061.

[181] Zheng, G., Awadallah, A. H., and Dumais, S. T. (2019). Meta label correction for learning with weak supervision. *CoRR*, abs/1911.03809.

[182] Zucchet, N., Schug, S., Von Oswald, J., Zhao, D., and Sacramento, J. (2022). A contrastive rule for meta-learning. *Advances in Neural Information Processing Systems*, 35:25921–25936.