Foundation Model Engineering: Engineering Foundation Models Just as Engineering Software

DEZHI RAN, Key Lab of HCST (PKU), MOE; School of Computer Science, Peking University, China
MENGZHOU WU, School of EECS, Peking University, China
WEI YANG, University of Texas at Dallas, United States
TAO XIE[*], Key Lab of HCST (PKU), MOE; School of Computer Science, Peking University, China

By treating data and models as source code, Foundation Models (FMs) become a new type of software. Mirroring the concept of software crisis, the increasing complexity of FMs makes FM crisis a tangible concern in the coming decade, appealing for new theories and methodologies from the field of software engineering. In this article, we outline our vision of introducing FM engineering, a strategic response to the anticipated FM crisis with principled engineering methodologies. FM engineering aims to mitigate potential issues in FM development and application through the introduction of declarative, automated, and unified programming interfaces for both data and model management, reducing the complexities involved in working with FMs by providing a more structured and intuitive process for developers. Through the establishment of FM engineering, we aim to provide a robust, automated, and extensible framework that addresses the imminent challenges, and discover new research opportunities for the software engineering field.

CCS Concepts: • **Software and its engineering** → **Software development techniques**; **Software configuration management and version control systems**; **Software maintenance tools**; • **Information systems** → *Data management systems*; • **Computing methodologies** → *Machine learning approaches.*

Additional Key Words and Phrases: Software Engineering, Artificial Intelligence, Machine Learning, Foundation Model, Data Management, Model Management, Version Control Systems

ACM Reference Format:
Dezhi Ran, Mengzhou Wu, Wei Yang, and Tao Xie. 2025. Foundation Model Engineering: Engineering Foundation Models Just as Engineering Software. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (February 2025), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Overview, Motivation, and Aims

Foundation Models [5] (in short as FMs) are becoming a new type of software. An analogy between traditional software and FMs is depicted in Figure 1, highlighting the parallel roles of their core components. In FM development, *data* and *models* play a critical role akin to source code in conventional software development. Developers curate datasets, such as sets of example input-output pairs, to articulate the specifications for the desired FM. They then choose an appropriate

[*]Tao Xie is the corresponding author.

Authors' Contact Information: Dezhi Ran, Key Lab of HCST (PKU), MOE; School of Computer Science, Peking University, Beijing, China, dezhiran@pku.edu.cn; Mengzhou Wu, School of EECS, Peking University, Beijing, China, wmz@stu.pku.edu.cn; Wei Yang, University of Texas at Dallas, Dallas, United States, wei.yang@utdallas.edu; Tao Xie, Key Lab of HCST (PKU), MOE; School of Computer Science, Peking University, Beijing, China, taoxie@pku.edu.cn.

ACM 1557-7392/2025/2-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

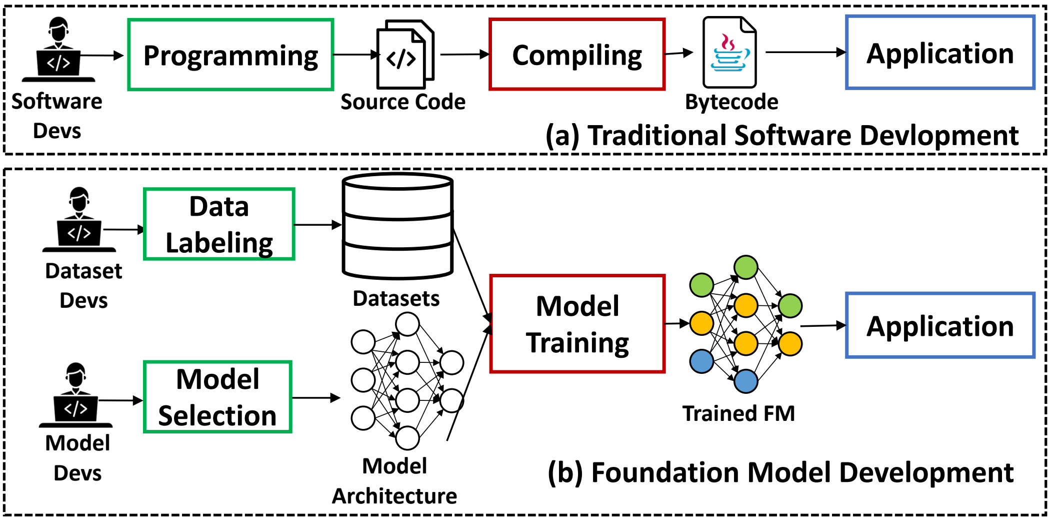

Fig. 1. An analogy between the development of traditional software and foundation models. In traditional software development, source code is manually written by human developers. The source code (e.g., cpp files) is compiled into an executable binary to perform specific tasks. In foundation model development, "source code" typically consists of two main components: 1) the dataset, which outlines the desired behavior, and 2) the neural network architecture, which provides a basic structure for the model, although many specifications (such as the weights) remain to be determined. Through the training process, the dataset is "compiled" into the final foundation model, which is similar to the compiled binary of traditional software.

network architecture and employ model training techniques such as backpropagation [64] to effectively "compile" the dataset and network architecture into the targeted FM.

Given the analogy between FMs and traditional software coupled with the increasing complexity of FM development (depicted in Figure 2), we envision that an FM crisis, mirroring the concept of the software crisis [11], will surface as a tangible concern over the coming decade. This potential crisis can be evidenced across four major aspects:

- **Increasing FM complexity.** Since the invention of Transformer [72], the complexity of FMs (represented by the number of parameters) has been increasing at an exponential rate. This growing complexity raises significant challenges not only in training and managing these FMs but also in managing the datasets that underpin them. Specifically, as datasets grow in size to unprecedented levels, the tasks of cleaning, labeling, and managing data at such a scale become increasingly challenging.
- **Continuous FM evolution.** FMs are in a state of continuous and fast evolution, driven by the integration of new data, the emergence of new requirements, and the implementation of bug fixes. For example, the average time between updates for OpenAI's models is 2 weeks [53], more frequent than that of Linux, whose mainline kernels are updated every 9-10 weeks [40]. Beyond the updates to the base FM, developers often finetune these models within their specific application domains. When the base FM updates, particularly security-related fixes, it necessitates a corresponding update to the finetuned FMs. It is challenging to efficiently and cost-effectively manage the evolution of these finetuned FMs, catching up with the base model updates while maintaining their domain-specific enhancements.

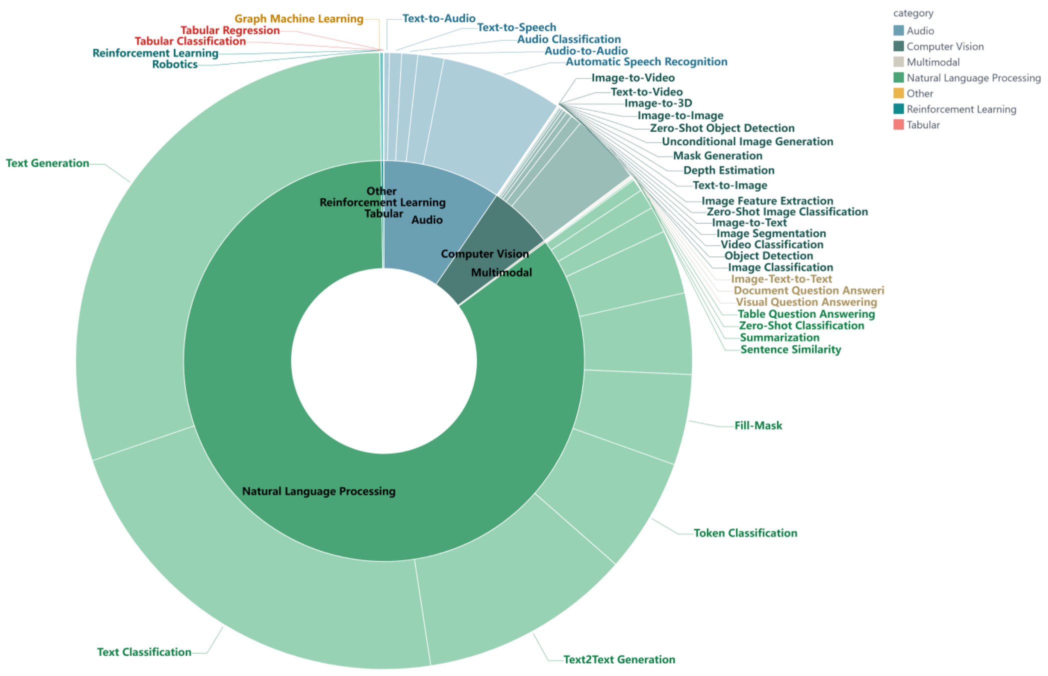

Fig. 2. Diverse customization demand. The staggering array of models, datasets, and AI applications available on Hugging Face [16], featuring 254,871 models with the Transformer architecture, 127,462 datasets, and over 170,000 AI app demos, vividly illustrates the diverse customization demand of FMs.

- **Diverse customization demand.** The era of FMs is characterized not only by their technological advancements but also by the broad spectrum of customization demands. On Hugging Face [16], there are 254,871 models with the Transformer architecture, 127,462 datasets, and over 170,000 AI app demos across various application domains. This diverse ecosystem of models, datasets, and applications is a clear indication that the future of FMs lies in their ability to be customized. During the customization, ensuring data privacy, model interpretability, and bias elimination requires innovations of data management and model management.
- **Multi-stakeholder collaboration.** The development of FMs involves multi-stakeholder collaboration among data scientists, model developers, and application domain experts. This collaboration introduces significant hurdles, primarily due to differing objectives, terminologies, and methodologies across disciplines. The specialized language used by data scientists and engineers may not align with the domain-specific knowledge and practical insights of application domain experts, potentially leading to misunderstandings and delays in project timelines. Innovative techniques are needed for minimizing the barriers posed by interdisciplinary work and fostering effective collaboration.

Inspired by the role that software engineering has played in addressing the software crisis [11], this article outlines our vision of *FM engineering* (depicted in Figure 3), a strategic response to the anticipated FM crisis. FM engineering aims to mitigate potential issues in FM development and application through the introduction of declarative, automated, and unified programming interfaces for both data and model management, reducing the complexities involved in working with FMs by providing a more structured and intuitive process for developers. These are three key components of our proposed FM engineering:

- **Data management with weak supervision.** Recognizing the importance of high-quality data in training effective FMs, we advocate for advanced data management strategies that leverage weak supervision. This approach allows for the efficient labeling and curating of vast datasets by combining limited amounts of labeled data with large quantities of unlabeled data, using algorithms to infer labels and improve data quality. This approach significantly reduces the time and resources required for data preparation.
- **Model management with workflows and continuous integration.** Effective model management is essential for the scalable and sustainable development of FMs. We envision the implementation of workflows that encompass model development, training, evaluation, and deployment processes, automating routine tasks and ensuring optimal practices. We also envision a distributed version control system like Git [43] to track the update of FMs, manage model branches, and resolve conflicts of model updates.
- **Programmatic FM development with declarative specifications.** To further simplify the engineering of FMs, we envision unified and declarative APIs that abstract away the underlying complexities of model and data management. These APIs allow developers to specify what they want to achieve in a high-level language, without needing to provide detailed instructions on how to accomplish these tasks. This way not only makes the process more accessible to a wider range of users, including those with less technical expertise but also accelerates the development cycle by enabling quicker iterations and refinements of models.

Through the establishment of FM engineering as envisioned in this article, we aim to provide a robust, automated, and extensible framework that addresses the imminent challenges and opportunities presented by the rapid advancement of FMs. By equipping developers with the tools and methodologies to effectively and efficiently manage data and models, we can enhance productivity and responsible use of these powerful FMs, and discover research opportunities for the software engineering community in the FM era. In summary, our aims are as follows:

- envision FM engineering for resolving the potential “FM crisis” in the next decade;
- architect FM engineering in terms of data management, model management, and declarative programming interfaces;
- research on tools, techniques, and methodologies for improving and automating FM engineering.

2 Envision of FM Engineering

2.1 Overview of FM Engineering

Figure 3 presents the overview of the envisioned Foundation Model Engineering (in short as FME), which manages resources, abstracts common operations, and provides APIs for related developers to engineer data and model in declarative ways.

Data management. In the landscape of FME, data management emerges as a key element. The ease with which machines generate massive volumes of data presents a unique challenge, especially in ensuring full coverage across enormous public or private information sources where collection complexities multiply. The criticality of data relevance cannot be overstated; amidst the deluge, distinguishing valuable data for analysis and decision-making becomes crucial. Moreover, the effort to process and refine this raw data into a form that is both accessible and actionable is a formidable task that FME tackles head-on.

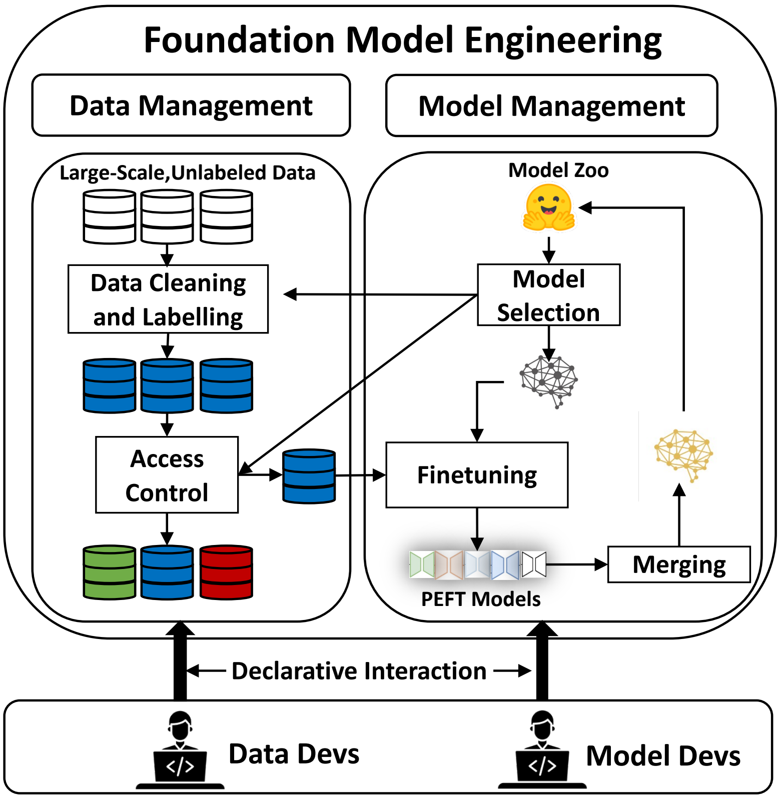

Fig. 3. Overview of Foundation Model Engineering.

At the core of FME's strategy is the fundamental principle that acquiring the appropriate data is critical to the success of the entire operation. It is widely recognized that simple models built on well-curated datasets often surpass their complex counterparts that are fed skewed or incomplete data. An integral part of this approach includes rigorous auditing and inspection of data pipelines, safeguarding data integrity and ensuring that processing aligns with predefined goals and compliance requirements.

Once raw data is collected, the next phase involves transforming this data into meaningful signals within FME. This transformation is a multi-step procedure that commences with data wrangling for cleaning and structuring, followed by aggregation to distill summary insights. Anomaly detection algorithms sift through to highlight irregularities, while pattern matching and linear regression inform on current trends and future directions. The process reaches its end with the deployment of advanced machine learning models that extract complex patterns and forecasts, thereby deepening the comprehension of the data's story.

FME maintains the data assets for model training and finetuning. Data developers can declare data labelling and cleaning functions with high-level intention descriptions and weak supervision, such as specifying exemplar data-label pairs. FME takes the schema as input, selects an appropriate model for parsing and understanding the schema (typically a code generation model or SQL generation model), and generates data labeling or cleaning functions to perform the data labelling or cleaning at scale. In addition to data labeling, FME also implements access control to data assets to assure that the data access conforms to legal policies as well as user intentions.

Model management. FME champions a novel paradigm for machine learning, paralleling the collaborative dynamism of open-source software development. This initiative seeks to transform the lifecycle of FMs from static entities to evolving constructs, continuously refined through community contributions. Unlike traditional open-source software, which thrives on collective inputs and evolution, FMs often see their development halt post-release. To bridge this gap, FME is cultivating a culture where machine learning models are not just released but are actively developed, enhanced, and adapted through collaborative efforts.

Central to this culture shift is the strategic facilitation of efficient change communication and contribution integration, steering clear of the impracticalities of transferring voluminous parameters (a characteristic of contemporary models). Leveraging insights like Fisher information [45], FME focuses on pinpointing and updating specific subsets of parameters. This targeted approach enables substantial performance enhancements without the heft of large-scale data transfers.

In the spirit of collaborative software engineering, the exploration of merging contributions from disparate sources stands as a testament to the power of collective intelligence. By integrating the diverse expertise of independently crafted models, FME paves the way for new capabilities and amplified performance in specialized tasks.

The push toward modularity in machine learning is informed by the "mixture of experts" architectural paradigm [14, 50, 78, 80], where specialized sub-networks synergize through adaptive routing, facilitating the backpropagation of discrete choices. Such an arrangement empowers models to assimilate domain-specific knowledge with unprecedented efficiency.

Lastly, to facilitate the version control and integration of updates, the concept of git for models is being developed [32]. This system aims to track parameter updates, support efficient merging, and integrate seamlessly with existing workflows, marking a significant step toward a more collaborative machine learning ecosystem.

FME also automates the model finetuning process, where developers specify tasks and declare accessible data assets. The system intelligently selects suitable models, identifies finetuning data from the available pool, and executes finetuning with auto-adjusted hyperparameters. Upon completion, FME merges the redundant models, resolves conflicts during merging, and returns the merged model for subsequent usage.

2.2 Data Management

As shown in Figure 4, we envision an integrated environment and declarative interfaces that support data management with minimal human efforts. Data is the user interface to "program" FMs, and a user-friendly interface is expected to fulfill four major requirements. First, FMs are trained on vast datasets, comprising billions of words and documents from the Internet, books, articles, and more. The quality, diversity, and size of the training data directly affect FMs' ability to understand and perform tasks ranging from dialog systems to code generation. Second, labeled data are required to adapt FMs to specific downstream tasks or non-functional properties such as human preference alignment [54]. Third, data sourcing, access control, and unlearning are crucial for data management to conform to legal policies and user-privacy requirements. Finally, the goal is to fulfill the preceding requirements in an automated streamline with minimal human efforts. To fulfill the preceding requirements, we envision that the data management of FME consists of the following four modules.

Data cleaning. The data cleaning module, critical in ensuring data integrity and quality for FMs, embraces a high-level, declarative framework. Within this framework, users specify the desired characteristics of clean data, guiding the module to automatically pinpoint and rectify inaccuracies,

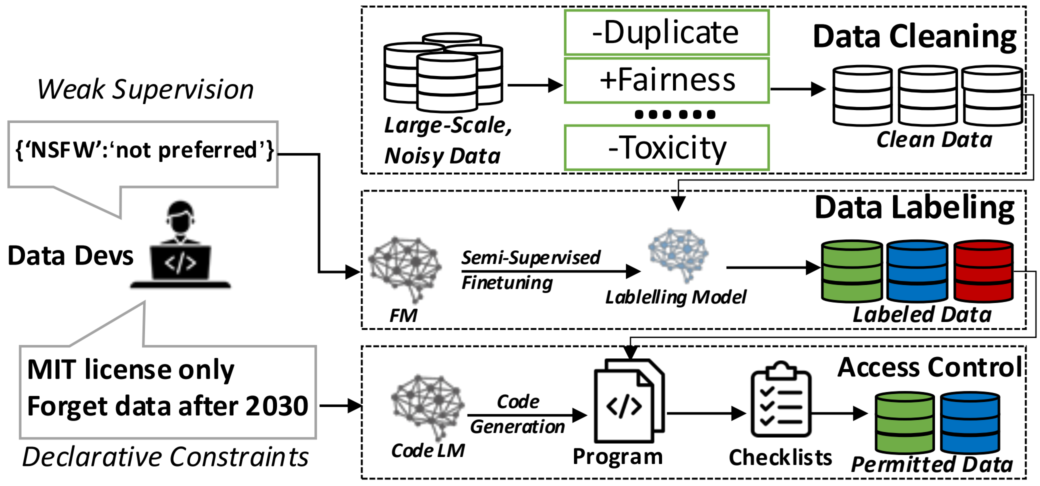

Fig. 4. Overview of Data Management.

inconsistencies, and redundancies in the datasets according to these specifications. The module is enriched with an interactive feedback loop. This crucial feature allows users to review and validate the automated cleaning actions undertaken by the system, facilitating an iterative refinement process. Through this dynamic interaction, users can fine-tune the cleaning criteria, ensuring that they resonate with the unique aspects of the data and their specific needs. By employing automated tools and algorithms, this module can detect anomalies, filter out irrelevant or sensitive information, and standardize data formats. This step is crucial for reducing noise in the training data, thereby enhancing FM's learning efficiency and effectiveness in understanding complex language patterns and generating coherent outputs.

Data labelling. Data labelling focuses on annotating the training data with informative tags or labels that define the context or the desired outcome of the data. This module is particularly important for supervised learning tasks where the FM needs to recognize patterns or generate responses based on specific inputs. Leveraging automated strategies of data labelling with human specifications establishes high-level rules or heuristics, facilitating the automatic creation of labels over large datasets. This approach is strengthened by incorporating specialized knowledge through labeling functions and using weak supervision techniques to utilize a broad range of information sources for deducing labels. In addition, the module iteratively enhances label accuracy by leveraging model-driven insights to rectify initial ambiguities or errors introduced by heuristic rules or noisy labels. This module enables the customization of FMs for specialized applications, from sentiment analysis to personalized content creation, by aligning the resulting model's outputs with human preferences and task-specific requirements.

Access control. Data sourcing must comply with copyright laws, privacy regulations, and ethical standards. The use of publicly available data, proprietary datasets, and user-generated content requires careful examination. Access control mechanisms are implemented to manage who can view or use the data, ensuring that only authorized users or systems have the ability to access or modify the datasets. By incorporating robust authentication, authorization, and auditing processes, the access control module safeguards sensitive information and prevents unauthorized data breaches, thereby fostering trust in FMs' development and deployment processes.

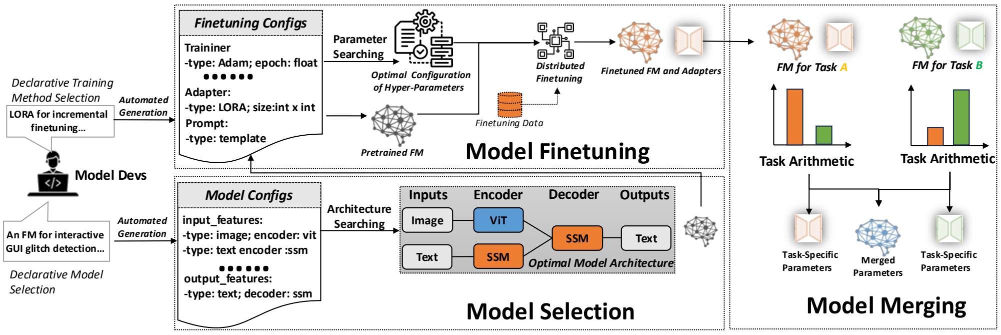

Fig. 5. Overview of Model Management.

Weak supervision. The weak supervision module aims to reduce the reliance on extensively labeled datasets by utilizing less-precise labels that can be generated more easily or derived from heuristic rules and external knowledge sources. This approach allows for the rapid scaling of training data while managing resource constraints and minimizing manual labelling efforts. Through advanced algorithms and models that can learn from weakly supervised data, this module supports the efficient training of FMs across a broader range of tasks and domains, accelerating the resulting model's adaptability and performance improvement.

2.3 Model Management

As shown in Figure 5, we develop an integrated environment and distributed version control system that supports the development, evolution and deployment of FMs.

Nowadays, model updates rarely start from retraining from scratch but instead involve incremental fine-tuning where only a small fraction of parameters are changed. Considering that different applications may finetune their own FMs from the same pretrianed FM, maintaining separate FMs for different applications encounters similar problems as code-cloning problems common in traditional software engineering, reducing software reliability and maintainability. Given that different users have different fine-tuning data, leading to different or even conflicting parameter updates, maintaining these FMs can be challenging for developers to manually check and update the FMs.

Inspired by Git [43], we envision a distributed version control system of FMs (shown in Figure 6), providing platform support for the community-driven development, evolution, and continuous integration of FMs. The system offers a unified abstraction of software development and version management for both new applications and the supporting platform.

Model selection. The model selection module empowers users to identify the most suitable FMs for their specific applications based on performance benchmarks, compatibility, and previous usage outcomes. For example, the Composition of Experts (CoE) [65] is used in model selection to aggregate multiple specialized models to improve overall performance and accuracy. For example, in a CoE system, there could be distinct expert models in language understanding, image recognition, and sentiment analysis. When faced with a complex task that involves understanding text within images and gauging sentiment, the CoE system would select and combine the outputs of these expert models. This modular approach allows for targeted fine-tuning of each expert model, ensuring that the collective output is both accurate and efficient in handling the task at hand.

Model finetuning. The model finetuning module offers a flexible and user-friendly toolkit for customizing FMs to meet the unique demands of diverse applications. It simplifies the process of

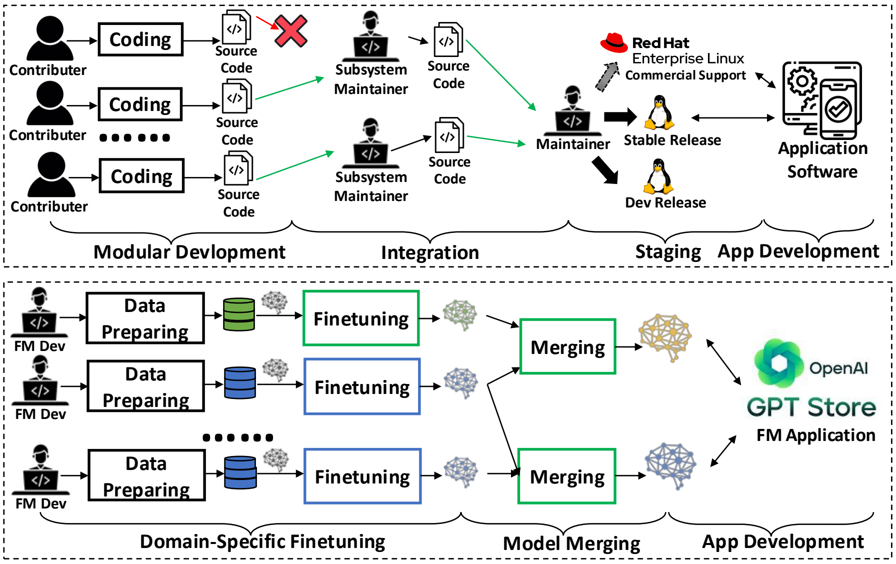

Fig. 6. Envisioned FM Update Process with an Analogy with Linux Update Process.

applying incremental updates, adjusting parameters, and integrating new data, thereby enabling personalized model optimization without the need for extensive machine learning expertise. The finetuning module provides an intuitive toolkit, enabling users to tailor FMs for diverse needs with ease. It supports collaborative, large-scale, distributed learning environments, where contributors work on separate data without sharing, ensuring data privacy and ownership while optimizing model performance through a central minimal-computation repository. This distributed setup allows for parallel training tasks on shared servers, maintaining parameter ownership and preventing task interference, streamlining the path to personalized model optimization.

Model merging. The process of model merging after finetuning involves a collaborative and incremental approach, similar to practices in open-source software development. Addressing the challenges of divergent fine-tuning efforts, the model merging module incorporates sophisticated algorithms to harmonize changes from multiple sources. This approach allows for efficient communication of updates between contributors and a central repository, focusing on updating only a selected subset of parameters based on their significance, as determined by measures like Fischer information [45, 68]. This approach ensures that updates are manageable in size, reducing communication costs and complexity. The ultimate goal is to combine the strengths of independently trained models, preserving the benefits of each model, and to enhance the overall performance and capabilities of the merged model in a distributed and collaborative learning environment. Achieving this goal ensures consistency, mitigates conflicts, and maintains the integrity of FMs across different applications, significantly reducing the maintenance burden and promoting collaborative improvements.

Model deployment. The model deployment module streamlines the process of rolling out updated or newly finetuned FMs into production environments. The deployment process begins with user requests, which are systematically queued. These requests are then managed by a deployment setup, often running on a Kubernetes environment, where an FM is loaded into memory within containers organized into pods. Within this deployment, there are two primary types of model weights: the base model weights, which form the core parameters of the FM, and adapter weights, which are a smaller set of parameters that allow for model fine-tuning.

One of the central challenges in deploying fine-tuned FMs is the significant resource requirements, particularly in terms of GPU usage. Each new user or task traditionally necessitates a new pod and GPU, leading to potentially excessive resource consumption. However, a crucial insight is that most of the model weights across different deployments remain identical, with variations primarily in the adapter weights. This realization opens up possibilities for sharing the base model across multiple adapters, thereby optimizing resource use.

Addressing this challenge, we envision a system with three innovative techniques. First, it employs dynamic loading of adapter weights, allowing the system to serve multiple user requests by loading only the necessary adapter weights alongside the base model weights into memory. This technique significantly reduces the need for additional resources per user or task. Second, the system incorporates a multi-tier weight cache to manage adapter weights efficiently. This cache includes a GPU cache for actively used adapters, a CPU cache for adapters awaiting activation, and an idle tier for adapters not currently in use, with their weights stored on ephemeral disk storage for potential future requests.

Finally, another key innovative technique that we can adopt here is continuous multi-adapter batching, an extension of the continuous batching concept [59]. This technique allows the system to process requests from multiple adapters together in the same batch, significantly improving throughput and efficiency. The batching algorithm central to this system prioritizes adapters based on request timestamps and employs a cycle-time parameter to manage the swapping of adapters in and out of the active set, striking a balance between throughput and latency. The model deployment module ensures smooth transition, minimizes downtime, and facilitates continuous delivery, allowing developers and users to leverage the latest advancements with ease and confidence.

Together, these techniques underlie a comprehensive ecosystem that not only simplifies the management of FMs but also accelerates the pace of innovation, fostering a collaborative and dynamic environment for the advancement of intelligent applications.

3 State of the Practice

3.1 Application Scenarios for Foundation Models

Foundation Models (FMs) are increasingly being deployed across a wide range of application scenarios [5]. In natural language processing [6], FMs power advanced conversational agents [52], text summarization tools [55, 71], and translation systems [4, 17]. In the field of computer vision, FMs are applied to tasks like object detection [61, 77], image classification [13, 23], and even complex activities like medical image analysis [84]. Moreover, FMs are finding applications in industries such as finance [85], where FMs are used for risk assessment [21] and predictive modeling [73], and in healthcare, where they contribute to personalized medical assistance [1, 82] and drug discovery [2, 31]. Each application scenario presents unique requirements and challenges, which must be carefully considered during the FM development process to ensure that the models are tailored to meet specific needs effectively.

3.2 Data-Centric Machine Learning

Kaplan et al. [33] reveal that improving model architectures usually offers limited benefits. In contrast, the efficacy of data utilization is becoming the cornerstone of advancing model performance given the increasing dataset scale enabled by self-supervised learning techniques [10, 60].

Data cleaning. Data cleaning involves addressing errors, duplications, and incompleteness in datasets by modifying, adding, or deleting data. Holoclean [63] utilizes a variety of techniques including heuristic rules (such as integrity constraints), external knowledge, and quantitative statistics to integrate multiple data sources into a probabilistic model, identifying and correcting errors in datasets. Picket [42] employs self-supervised deep learning models to identify and remove corrupted data without the need for human supervision. Neutatz et al. [51] find that the benefits of data cleaning largely depend on the application, leading to the proposal of end-to-end, application-driven, holistic data cleaning approaches. ActiveClean [36] combines data cleaning with active learning, prioritizing the cleaning of data that could potentially impact model performance in specific application domains.

Data programming with weak supervision. The need for large labeled datasets to optimally train modern machine learning models presents a major bottleneck, due to the high costs and time needed for expert manual annotation. In response, active learning strategies [66] streamline this process by selectively engaging experts to label data of maximal utility—such as instances at the fringes of classification models—thereby amplifying model efficacy with diminished input. Parallelly, semi-supervised learning [70] capitalizes on a modest quantity of labeled data supplemented by unlabeled data to enhance model accuracy, effectively economizing on the need for extensive labeled datasets. Weak supervision approaches harness cost-effective techniques for gathering lower-quality labeled data through avenues like Crowdsourcing [34], Distant Supervision [49], or heuristic rules, markedly alleviating the reliance on manual labeling. The Snorkel system [62] enables users to employ labeling functions that encapsulate heuristic techniques, and combines different weak supervision sources to generate a probabilistic distribution of labels.

3.3 Incremental Model Training

Parameter-Efficient Fine-Tuning (PEFT). The extensive parameter set of FMs renders their fine-tuning both computationally expensive and storage-intensive. PEFT approaches [46] offer a solution by fine-tuning a fraction of their parameters. The compact nature of specialized PEFT modules facilitates their dissemination within the community, as evidenced by the availability of over 20,000 adapters on the Hugging Face Model Hub, all of which are based on PEFT.

PEFT approaches are predominantly categorized into three distinct types. First, *additive approaches* entail the integration of additional parameters into the original FM architecture, with the fine-tuning process focusing exclusively on these new parameters. Notably, Houlsby et al. [26] introduce fully-connected networks as adapter modules within the transformer architecture, after the attention and Feed-Forward Network (FFN) layers. Prompt tuning [37] and prefix tuning [38] incorporate task-specific vector sequences at the input layer and throughout various layers of the FM, respectively, with fine-tuning achieved through the adjustment of these vector parameters. Second, *selective approaches* involve the selective training of a subset of the FM's parameters. BitFit [79] fine-tunes only the bias parameters, while DiffPruning [20] employs an L0-norm to train a sparse weight matrix. Freeze and Reconfigure [75] and FishMask [68] identify and train crucial model parameters based on L1-distance and Fisher information, respectively. Third, *reparametrization-based approaches* modify the original model's parameter matrix into a more tractable low-rank format for training purposes. Intrinsic SAID [3] utilizes the FastFood transform for reparameterizing model weight updates. LoRA [27] decomposes the weight matrix updates into products of low-rank matrices, with KronA [15] employing Kronecker product for matrix factorization. AdaLoRA [83] adopts Singular Value Decomposition (SVD) for parameter matrix decomposition, prioritizing resources based on the significance of different weight matrices. Additionally, innovative integrations such as SparseAdapter [24], MAM Adapters [22], UniPELT [47], Compacter [35], and S4 [7] amalgamate various PEFT approaches to enhance efficiency and adaptability.

Modular training for multi-task learning. Ilharco et al. [29] introduce the concept of a task vector to delineate the shifts within the model's parameter space consequent to fine-tuning for a specific task. Ilharco et al. [29] further elucidate that performing arithmetic operations on this task vector facilitates the processes of forgetting, constructing multi-task models, and task analogies. In the quest to refine model merging capabilities by leveraging the task vector, Matena and Raffel [48] apply Fisher information weighting, and Jin et al. [30] frame the challenge as an optimization quandary and resolve it via linear regression, while Yadav et al. [76] mitigate interference by eliminating superfluous parameters and reconciling symbol discrepancies. Choshen et al. [8] and Don-Yehiya et al. [12] engage in iterative fine-tuning of the FM across diverse tasks, subsequently averaging the weights to enhance the FM. Further, an amalgamation of PEFT approaches with model merging strategies is explored [9, 28, 44, 56, 58, 81]. Ponti et al. [58] posit that each task is linked to a spectrum of skills, with each skill mirrored by an adapter, and devise a routing function to allocate skills per task. AdapterSoup [9] employs task textual similarity and clustering techniques to identify auxiliary tasks conducive to the target task, facilitating the merger of pertinent adapters. LoRAHub [28] adopts a gradient-free optimization strategy to fine-tune LoRA model merging, guided by few-shot examples of the target task.

A noteworthy trend in recent research [14, 50, 78, 80] is the development of mixture-of-experts models to tackle the multi-task conundrum. Within such models, a selective activation of a subset of experts is triggered at each layer contingent on the input, thereby focusing inference and training efforts solely on the activated experts. This approach has been shown to yield superior performance, particularly when models are extended to tasks beyond their initial training scope.

4 Research Opportunities

To fulfill our vision of Foundation Model (FM) engineering, numerous research opportunities arise in three major categories, including declarative FM engineering, fine-grained data management, and automating model management.

4.1 Declarative FM Engineering

Declarative FM engineering represents a paradigm shift in the development of Large Language Models, emphasizing the specification of what a model should achieve rather than how it achieves the specification. This approach, rooted in the principles of declarative programming, offers a more intuitive and efficient methodology for designing, training, and deploying FMs. It invites a wealth of research opportunities aimed at simplifying the complex process of FM engineering, making it more accessible and adaptable to a broader range of applications and developers.

FM requirements engineering. The requirements engineering of FMs plays a crucial role in guiding FM development and deployment. First, in terms of data requirements engineering, it is essential to identify and curate datasets that are not only large and diverse but also aligned with the specific tasks that the FM is expected to perform. Second, model requirements engineering focuses on defining the specifications and capabilities that the FM must possess to meet its intended use cases. For example, in safety-critical applications, the model may need to meet stringent accuracy and reliability standards, while in customer-facing applications, user experience factors like response time and interpretability might be prioritized. Research on FM requirements engineering could focus on creating methodologies and pipelines for an effective understanding of FM application goals, and an efficient selection of FMs.

High-level model specification languages. For FM engineering, developing high-level, domain-specific languages that allow developers to specify the desired outcomes, constraints, and behaviors of the model in an abstract manner. Research in this area could focus on creating intuitive syntax and semantics that encapsulate the complexities of neural network architectures, training procedures, and data processing pipelines.

Automated model synthesis. Building on high-level specifications, automated model synthesis involves research into algorithms and systems capable of translating these abstract descriptions into concrete, optimized FM architectures. Such systems include selecting appropriate neural network components, configuring layers and connections, and determining optimal training strategies based on the specified objectives and constraints.

Constraint-based optimization. Constraint-based optimization investigating techniques for incorporating various types of constraints (e.g., performance, fairness, privacy) directly into the FM training process in a declarative manner. This research area would explore optimization techniques that can balance multiple objectives and adhere to specified constraints, ensuring that the resulting models align with ethical guidelines and application-specific requirements.

Resource-aware engineering. The development of FMs significantly differs from traditional software development due to the substantial infrastructure requirements, particularly in terms of computational resources. Training and fine-tuning FMs necessitate large-scale computational power, often involving clusters of GPUs or NPUs/TPUs to handle the extensive data and complex calculations required [25, 69]. This disparity not only creates a barrier to entry for smaller entities but also limits the diversity of contributors to the FM development process, potentially leading to biases in the models due to a lack of varied perspectives.

The infrastructure challenges in FM development open up significant research opportunities aimed at democratizing access to these resources. One research direction is the exploration of more efficient model training techniques, such as model pruning [41, 86], quantization [57], and knowledge distillation [19, 57], which can reduce the computational burden without compromising model performance. Another research direction is the development of distributed and collaborative training frameworks [39] that allow multiple smaller entities to pool their computational resources, making FM development more accessible.

Verification and validation. Given the abstract nature of declarative specifications, developing robust verification and validation techniques is crucial to ensure that the synthesized FMs faithfully represent the intended outcomes and adhere to all specified constraints. These techniques include formal verification techniques to prove the properties of the models and empirical validation techniques to evaluate their performance and behavior in real-world scenarios.

4.2 Fine-grained Data Management

Granular permissions. Developing systems that enable granular control over who can access specific datasets or parts of datasets is crucial. Such control includes defining roles and permissions at a detailed level, and allowing for precise management of data access based on the user's role, the nature of the project, and the sensitivity of the data. Research into models that can dynamically adjust permissions in response to changing project needs or data sensitivity levels could significantly enhance data security and governance.

Decentralized access control. With the rise of decentralized technologies, investigating decentralized access control models, such as those based on blockchain, could offer new ways to manage data assets securely and transparently. These models could provide immutable, verifiable logs of data access and changes, enhancing trust and compliance.

Automated compliance checks. Given the complex web of data protection laws globally, developing automated systems for compliance checks during data access can help organizations navigate legal requirements more efficiently. Research into AI-driven compliance advisors that can interpret and apply legal rules in real-time during data access decisions could greatly reduce the burden of legal compliance.

Secured data management. Given that FMs rely on vast amounts of data (for training and fine-tuning), which may include sensitive information such as personally identifiable information

(PII) and confidential business data, safeguarding this data throughout its lifecycle is critical. To achieve the safeguarding, multi-layered access control mechanisms can be implemented to restrict unauthorized access and ensure that data remains encrypted during transmission between users and systems. Additionally, automated compliance check systems can monitor data access and processing in real-time, ensuring that all operations adhere to relevant legal regulations, such as GDPR [74]. Weak supervision techniques can also reduce the need for manual intervention in data labeling by leveraging broader information sources to automatically generate labels, thus enhancing data utilization efficiency while maintaining data privacy and security [5, 62].

4.3 Automating Model Management

Continuous evolution of FMs. The need to evolve FMs parallels the dynamic nature of open-source software development, emphasizing the critical requirement for robust infrastructure support for incremental, collaborative, and agile enhancement of FMs. FMs, much like any sophisticated software, may require updates for a variety of critical reasons: enhancing their performance through extended training or alternative data, rectifying problematic outputs such as noise or offensive content, and addressing privacy concerns related to memorized data. However, the current practice largely involves models remaining static post-release, awaiting replacement by a completely new version, rather than undergoing continuous refinement. This current practice contrasts with the evolutionary trajectory of open-source projects, like popular programming languages such as Python or Java. If these languages had remained unchanged since their initial release, they would lack many now-essential features and fixes that have been contributed by a diverse community of developers. These enhancements, often seamlessly integrated into existing codebases, are facilitated by a well-established ecosystem of development tools and practices, including version control, continuous integration, and package management. Adopting a similar framework for the development of FMs would not only enable their continual improvement but also ensure their relevance and utility in an ever-evolving technological landscape. Such a framework necessitates the establishment of standardized protocols for model updates, a transparent versioning system to manage changes, and comprehensive guidelines for adapting existing applications to updated models.

Version control systems of FMs. The success of the Linux operating system ecosystem can be largely attributed to its extensive array of both built-in and externally contributed libraries. Managing these libraries effectively (including their installation, removal, and dependency resolution), along with tracking developmental progress through version control, has been crucial to the success. Git, a distributed version control system that emerged from the Linux ecosystem, has been instrumental in fostering collaborative and parallel development, significantly accelerating the ecosystem's growth and evolution. While Git excels at managing code through line-level change tracking during operations like merging and rebasing, it faces new challenges when applied to natural language contents. Unlike source code, which follows strict syntactic rules, natural language expressions can vary significantly while maintaining semantic equivalence. For example, multiple developers might express the same concept using different phrasings, making it difficult to apply traditional version control approaches that focus on textual differences rather than semantic meaning. This challenge becomes particularly relevant as we see increased development of AI agents using natural language specifications. Traditional version control systems need to evolve to better handle these variations in natural language expressions while maintaining their ability to track and manage changes effectively.

Deployment security. Security of FMs is equally crucial, especially when they are deployed in critical tasks or handle sensitive information. Model security can be ensured through several measures, including access control and version control. Using distributed version control systems like

Git can effectively track model updates and changes, preventing unauthorized modifications [32]. Additionally, model security involves defending against adversarial attacks, which may manipulate a model's output by feeding it malicious input. To achieve model security, adversarial training and model validation techniques can be incorporated to enhance the model's resilience against such attacks [18]. Finally, regular security audits and performance evaluations of models help in identifying and addressing potential vulnerabilities, ensuring that the models remain secure and reliable as they evolve and are applied in various contexts [67].

5 Conclusion

In this article, we have envisioned foundation model (FM) engineering, aiming to streamline the development of FMs through innovative infrastructure software and methodologies. By simplifying data and model management while emphasizing automated, declarative interfaces, we envision a future where FMs are more accessible, efficient, and ethically developed. This approach not only promises to accelerate innovation in the machine learning field but also ensures that the profound benefits of FMs can be leveraged across various sectors, contributing positively to societal advancement. Moving forward, collaborative and conscientious efforts will be key to realizing the full potential of these technologies in a responsible and beneficial manner.

Acknowledgments

This work was partially supported by National Natural Science Foundation of China under Grant No. 623B2006 and Grant No. 92464301, an Amazon Trust AI Research Award, and the Tencent Foundation/XPLORER PRIZE.

References

[1] Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. 2023. Conversational health agents: A personalized LLM-powered agent framework. *arXiv preprint arXiv:2310.02374* (2023).

[2] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. 2024. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature* (2024), 1–3.

[3] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. 2020. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255* (2020).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[5] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *NIPS* 33 (2020), 1877–1901.

[7] Jiaao Chen, Aston Zhang, Xingjian Shi, Mu Li, Alex Smola, and Diyi Yang. 2023. Parameter-efficient fine-tuning design spaces. *arXiv preprint arXiv:2301.01821* (2023).

[8] Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. 2022. Fusing finetuned models for better pretraining. *arXiv preprint arXiv:2204.03044* (2022).

[9] Alexandra Chronopoulou, Matthew E Peters, Alexander Fraser, and Jesse Dodge. 2023. Adaptersoup: Weight averaging to improve generalization of pretrained language models. *arXiv preprint arXiv:2302.07027* (2023).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[11] Edsger W Dijkstra. 1972. The humble programmer. *Commun. ACM* 15, 10 (1972), 859–866.

[12] Shachar Don-Yehiya, Elad Venezian, Colin Raffel, Noam Slonim, Yoav Katz, and Leshem Choshen. 2022. Cold fusion: Collaborative descent for distributed multitask finetuning. *arXiv preprint arXiv:2212.01378* (2022).

[13] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[14] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *ICML*. PMLR, 5547–5569.

[15] Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650* (2022).

[16] Hugging Face. 2024. Hugging Face Datasets. https://huggingface.co/datasets

[17] Patrick Fernandes, Behrooz Ghorbani, Xavier Garcia, Markus Freitag, and Orhan Firat. 2023. Scaling laws for multilingual neural machine translation. In *ICML*. PMLR, 10053–10071.

[18] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[19] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *IJCV* 129, 6 (2021), 1789–1819.

[20] Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463* (2020).

[21] Md Morshadul Hasan, József Popp, and Judit Oláh. 2020. Current landscape and influence of big data on finance. *Journal of Big Data* 7, 1 (2020), 21.

[22] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366* (2021).

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[24] Shwai He, Liang Ding, Daize Dong, Miao Zhang, and Dacheng Tao. 2022. Sparseadapter: An easy approach for improving the parameter-efficiency of adapters. *arXiv preprint arXiv:2210.04284* (2022).

[25] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[26] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *ICML*. PMLR, 2790–2799.

[27] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[28] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. Lorahub: Efficient cross-task generalization via dynamic lora composition. *arXiv preprint arXiv:2307.13269* (2023).

[29] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2022. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089* (2022).

[30] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. 2022. Dataless knowledge fusion by merging weights of language models. *arXiv preprint arXiv:2212.09849* (2022).

[31] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[32] Nikhil Kandpal, Brian Lester, Mohammed Muqeeth, Anisha Mascarenhas, Monty Evans, Vishal Baskaran, Tenghao Huang, Haokun Liu, and Colin Raffel. 2023. Git-Theta: A Git extension for collaborative development of machine learning models. In *ICML*. PMLR, 15708–15719.

[33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[34] David Karger, Sewoong Oh, and Devavrat Shah. 2011. Iterative learning for reliable crowdsourcing systems. *NIPS* 24 (2011).

[35] Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *NIPS* 34 (2021), 1022–1035.

[36] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *VLDB* 9, 12 (2016), 948–959.

[37] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).

[38] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190* (2021).

[39] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[40] Inc Linux Kernel Organization. 2024. The Linux Kernel Archives. https://www.kernel.org/releases.html

[41] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).
[42] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2022. Picket: guarding against corrupted data in tabular data during learning and inference. *VLDBJ* 31, 5 (2022), 927–955.
[43] Jon Loeliger and Matthew McCullough. 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.".
[44] Xingtai Lv, Ning Ding, Yujia Qin, Zhiyuan Liu, and Maosong Sun. 2023. Parameter-efficient weight ensembling facilitates task-level knowledge transfer. In *ACL*. 270–282.
[45] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers. 2017. A tutorial on Fisher information. *Journal of Mathematical Psychology* 80 (2017), 40–55.
[46] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.
[47] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577* (2021).
[48] Michael S Matena and Colin A Raffel. 2022. Merging models with fisher-weighted averaging. *NIPS* 35 (2022), 17703–17716.
[49] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL*. 1003–1011.
[50] Mohammed Muqeeth, Haokun Liu, and Colin Raffel. 2023. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745* (2023).
[51] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2021. From cleaning before ML to cleaning for ML. *IEEE Data Eng. Bull.* 44, 1 (2021), 24–41.
[52] OpenAI. 2023. Introducing to ChatGPT. https://openai.com/blog/chatgpt
[53] OpenAI. 2024. ChatGPT — Release Notes. https://help.openai.com/en/articles/6825453-chatgpt-release-notes
[54] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *NIPS* 35 (2022), 27730–27744.
[55] Vaidehi Patil, Leonardo Ribeiro, Mengwen Liu, Mohit Bansal, and Markus Dreyer. 2024. REFINESUMM: Self-refining MLLM for generating a multimodal summarization dataset. In *ACL*. 13773–13786.
[56] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247* (2020).
[57] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).
[58] Edoardo Maria Ponti, Alessandro Sordoni, Yoshua Bengio, and Siva Reddy. 2023. Combining parameter-efficient modules for task-level generalisation. In *EACL*. 687–702.
[59] Predibase. 2023. Lorax: An Open Source Project by Predibase. https://github.com/predibase/lorax. Accessed: 2024-04-06.
[60] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[61] Dezhi Ran, Zongyang Li, Chenxu Liu, Wenyu Wang, Weizhi Meng, Xionglin Wu, Hui Jin, Jing Cui, Xing Tang, and Tao Xie. 2022. Automated visual testing for mobile apps in an industrial setting. In *ICSE-SEIP*. 55–64.
[62] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *VLDB*, Vol. 11. 269.
[63] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
[64] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.
[65] SambaNova. 2024. A Composition of Experts. https://sambanova.ai/technology/composition-of-experts.
[66] Burr Settles. 2009. Active learning literature survey. (2009).
[67] Dawn Song, Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramer, Atul Prakash, and Tadayoshi Kohno. 2018. Physical adversarial examples for object detectors. In *WOOT*.
[68] Yi-Lin Sung, Varun Nair, and Colin A Raffel. 2021. Training neural networks with fixed sparse masks. *NIPS* 34 (2021), 24193–24205.
[69] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[70] Jesper E Van Engelen and Holger H Hoos. 2020. A survey on semi-supervised learning. *Machine learning* 109, 2 (2020), 373–440.

[71] Tempest A van Schaik and Brittany Pugh. 2024. A field guide to automatic evaluation of LLM-generated summaries. In *SIGIR*. 2832–2836.

[72] A Vaswani. 2017. Attention is all you need. *NIPS* (2017).

[73] Frederic Voigt, Kai Von Luck, and Peer Stelldinger. 2024. Assessment of the applicability of large language models for quantitative stock price prediction. In *PETRA*. 293–302.

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

[75] Danilo Vucetic, Mohammadreza Tayaranian, Maryam Ziaeefard, James J Clark, Brett H Meyer, and Warren J Gross. 2022. Efficient fine-tuning of BERT models on the edge. In *ISCAS*. IEEE, 1838–1842.

[76] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. *NIPS* 36 (2024).

[77] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, et al. 2021. Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432* (2021).

[78] Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. 2023. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning. *arXiv preprint arXiv:2309.05444* (2023).

[79] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199* (2021).

[80] Fan Zhang, Duyu Tang, Yong Dai, Cong Zhou, Shuangzhi Wu, and Shuming Shi. 2022. SkillNet-NLU: A sparsely activated model for general-purpose natural language understanding. *arXiv preprint arXiv:2203.03312* (2022).

[81] Jinghan Zhang, Junteng Liu, Junxian He, et al. 2024. Composing parameter-efficient modules with arithmetic operation. *NIPS* 36 (2024).

[82] Kai Zhang, Yangyang Kang, Fubang Zhao, and Xiaozhong Liu. 2024. LLM-based medical assistant personalization with short-and long-term memory coordination. In *NAACL*. 2386–2398.

[83] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2022. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*.

[84] Shaoting Zhang and Dimitris Metaxas. 2024. On the challenges and perspectives of foundation models for medical image analysis. *Medical image analysis* 91 (2024), 102996.

[85] Huaqin Zhao, Zhengliang Liu, Zihao Wu, Yiwei Li, Tianze Yang, Peng Shu, Shaochen Xu, Haixing Dai, Lin Zhao, Gengchen Mai, et al. 2024. Revolutionizing finance with llms: An overview of applications and insights. *arXiv preprint arXiv:2401.11641* (2024).

[86] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).