E-Scan: Consuming Contextual Data with Model Plugins

Viktor Sanca[1,*], Anastasia Ailamaki[1,2,†]

[1]EPFL, Lausanne, Switzerland
[2]Google, Sunnyvale, USA

Abstract

Extracting value and insights from increasingly heterogeneous data sources involves multiple systems combining and consuming the data. With multi-modal and context-rich data such as strings, text, videos, or images, the problem of standardizing the data model and format for interchangeable use is further exacerbated by a non-uniform way of processing, extracting, and preserving content and context from the data. This makes the data movement, reuse, and exchange between different systems a non-composable, manual process. On the other hand, increasingly powerful and popular machine learning-driven data representation models map the input data into uniform high-dimensional vector embeddings for further processing, informed by particular models. However, using models is expensive, and the manual integration effort might exacerbate unnecessary costs.

Thus, we propose E-Scan, a contextual data exchange plugin for using, exchanging, and caching context-rich data. We outline the need for a common interface that separates the concerns and allows smooth and cost-effective data exchange. First, while vector embeddings are context-less, the model information is saved to preserve the context and preprocessing steps. Next, a lightweight vector engine caches and stores the uniform intermediate data representation in a lazy way to lower the transformation and data access, exchange, and retrieval cost. Finally, a pull-based interface allows uniform data consumption between components under a common plugin interface. This way, various context-rich data types are stored, processed, and exchanged in a standardized way while allowing plugin-based customization for subsequent context interpretation.

Keywords

Vector Data Management, Embeddings, ML for DB, Data Movement, Caching, Context-Rich Data, Composability

1. Introduction

Technological advances, the proliferation of the Internet, personal multimedia and sensor devices, and social media have changed data types and formats. While tabular, numerical, and generally relational data represent the backbone of many applications, the main task of data analytics is to provide and support timely, efficient, and declarative value extraction. Therefore, supporting novel and useful ways to process the data is a natural goal of modern analytics.

Machine learning methods are particularly dominant in extracting insights from context-rich data, which we analyze more deeply in a recent study on context-rich analytical engines and their future architectures [1], where the main takeaway is that different information sources, such as images, text, traditional relational data, and metadata, will interact in a complex and potentially ad-hoc analytical query. First of all, there are different machine learning frameworks such as Tensorflow [2] and PyTorch [3], which are specialized and optimized for machine learning workloads and represent a class of dataflow processing systems. These systems are useful for training and inference and will likely be components of a more complex system that requires coordination.

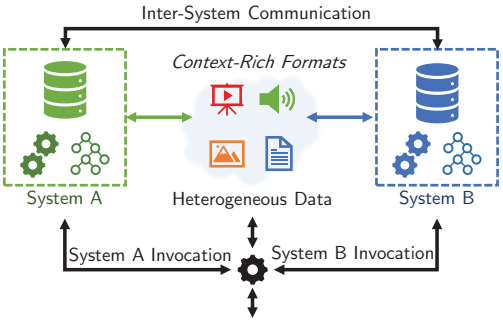

Figure 1: Complex system interactions for processing multi-modal data resemble polystore architectures.

On the other hand, data might come from specialized engines or object stores - images, videos, audio, and documents might be stored in separate engines. Finally, they might be combined with relational data in a complex hybrid ML-relational query plan to combine insights from the information coming from heterogeneous and multi-

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Second International Workshop on Composable Data Management Systems (CDMS'23), August 28 - September 1, 2023, Vancouver, Canada
*Corresponding author.
†Work done entirely at EPFL.
viktor.sanca@epfl.ch (V. Sanca); anastasia.ailamaki@epfl.ch (A. Ailamaki)
https://viktorsanca.com (V. Sanca)
0000-0002-4799-8467 (V. Sanca); 0000-0002-9949-3639 (A. Ailamaki)

modal data sources, for example, combining sentiment analysis, object detection, and performing joins with corresponding relational data.

No matter the implementation, as a monolithic system with internal communication or multiple standalone components and modules, the initial data is exchanged and transformed in a polystore-like fashion [4, 5]. While relational analytics come with strict schema and operator transformations for data compatibility, introducing multi-modality and model-driven transformation, the data movement and exchange process becomes manual and imperative, especially when using machine learning models that can perform many tasks. Therefore, the data, information, and task flow becomes an arbitrary process that is finally left to the end-user to optimize, as illustrated in Figure 1.

Consuming contextual data becomes complex not only for specifying the resulting schema and data movement but equally due to comparatively more expensive data processing when using models. This can result in higher computational resource requirements, increased latency, and higher monetary costs, especially in ad-hoc imperative settings where data that is not relevant might be processed, or data of frequent relevance might be unnecessarily processed multiple times, resulting in inefficient resource utilization.

Machine learning models for multi-modal tasks often have an expensive model-specific input data embedding step, followed by corresponding post-processing or operations performed directly on embeddings, such as similarity search. Conversely, this data might need to be mapped back to the original representation for the output or further analysis, requiring a link to the original object.

We consider these requirements and:

- describe model-driven processing, their associated costs, and several use cases in Section 2,
- outline interaction between the original objects and common intermediate, model-driven vector data representation in Section 3,
- propose a design for efficient and composable exchange of contextual data supporting multi-modal data and embedding-based models Section 4.

Our design proposal is named E-scan and aims to motivate a common *extensible* interface with new models and data formats that enables *efficient* contextual data exchange, caching, and lightweight processing behind a common plugin/connector-based interface, with a focus on vector *embeddings*.

2. Contextualized Data Consumption

A significant corpus of work in machine learning, especially in representation learning, is the key enabler of multi-modal and context-rich analytics. They transform the human-centric, context-rich data into machine-centric formats amenable to further automated processing. Tasks such as sentiment analysis, similarity search, object detection, translation, and data generation become possible - and we consider these tasks as parts of a more complex system.

While on their own, they are useful, such models can be combined with traditional analytics to perform multi-modal value extraction. For example, performing sentiment analysis on the text associated with an image that might contain certain objects on a retail website connected to transactions in a traditional RDBMS can provide new sources of insight or decision-making, utilizing the data already collected and stored across different systems.

2.1. Models for Multi-Modal Context-Rich Data

Before the advent of ML-driven data processing, large-scale analysis of contextual data often involved human-in-the-loop through crowdsourcing approaches such as Amazon Mechanical Turk[6], reCAPTCHA[7], or hiring domain experts. While useful in the early days of generating datasets for training the models, human-based analysis is slow, error-prone, and expensive for ad-hoc analytics.

Natural language processing has long been the study of representation learning, resulting in approaches such as word2Vec [8] or FastText [9] that allow operations such as context-string-similarity and classification, even for misspellings [10] and out-of-dictionary words. More complex approaches based on Transformer architecture [11] resulted in popular models such as BERT [12], or GPT-3 [13] and GPT-4 [14], that allowed more complex tasks such as translation and text generation. The change of complexity of the embedding and processing method has increased the computational (and monetary) cost of processing, but equally, the functionality that those models offer as part of the data processing pipeline and automated, machine-driven insights.

Furthermore, a rich research area in machine learning drives embedding models that support other context-rich data formats, equally transforming the input into subsequently processed embeddings, depending on the task and the architecture. Having specialized models for tasks allows multi-modal processing by selecting an appropriate method for the task. Models such as Segment Any-

thing Model (SAM) [15], ResNet [16], or Dall-E [17] can be used for model-driven image processing and generation. PANNs [18] or Whisper [19] are designed for audio processing. Finally, models trained on web-scale data exist as Foundation Models [20], that can be re-trained and adapted for a specific task and dataset without expensive re-building from scratch.

Using models, analytics evolve from joins and aggregations into more complex operations such as object detection, sentiment analysis, classification, and similarity operations. However, a potential issue is that the communication between the data storage, model specification, and processing using a particular model and framework are decoupled, and the interactions become increasingly complex, in the case as simple as what the model inputs should be, and what transformations and schema will the model output for subsequent processing. We will consider this in our system design.

2.2. The Cost of Model-Driven Embedding

Model-based analytics introduce complex and expensive operations for large-scale data processing and challenges in optimizing the cost and resources. Firstly, such operations often require computational resources like GPUs to instantiate the models and achieve desirable latency. However, some models might not be open-sourced, resulting in monetary costs that are increasingly high if models are not used frugally. We outline the pricing of different text embedding models of OpenAI in Table 1.

Table 1
Different text embedding costs (openai.com/pricing)

Model	In ($/1K tok.)	Out ($/1K tok.)
GPT-4 (8K context)	0.03	0.06
GPT-3.5-Turbo (4K)	0.0015	0.002
Davinci	0.03	0.12
Curie	0.003	0.012

If we consider the cost of processing single words only (a token) and an operation that embeds the words in an object or column store, performing this operation on a relatively modest data size of 1M tuples would result in a cost from $3 to $30 for that operation only in case of single words. This cost is likely higher for realistic scenarios over sentences or documents. We cannot discard the monetary and computational costs, and this cost should be amortized and invoked only when necessary.

Furthermore, it is unlikely to have 1M unique words, nor is their frequency equal or of the same interest. Embedding the same words once and then reusing this embedding motivates the need for caching mechanisms that we introduce in our plugin design.

A similar situation holds for other data formats regarding the cost Table 2, as formats such as images or audio might have more complex architectures and processing and, consequentially, pricing. We note that open-source models can be used, of similar characteristics, on repositories such as HuggingFace [21] or TensorFlow Hub [22]. Still, the cost of local or cloud resources remains for instantiating and running these models, depending on the particular cloud provider. Ideally, some processing can be avoided using caching, but this might not be immediately possible in the case of images or generative AI based on given data-driven prompts.

Still, analytical queries are typically selective, and not all data is of equal interest for analysis, which motivated prior research in lazy data ingestion using NoDB [23] architectures. In our case, this motivates pull-based model invocation where data is not eagerly embedded and processed but only on demand from the consuming/invoking operator. Allowing this process to happen in an ad-hoc, imperative setting in complex systems risks higher than necessary costs and longer latency for processing data, which would later be discarded or re-processed fully multiple times.

We present the caching and pull-based plugin design that aims to reduce the cost and resource requirements and reduce latency in Section 4.

3. Intermediate Data Representation

In this work, we focus on models which, at some processing phase, transform the context-rich data of various formats into embeddings. Embeddings are high-dimensional vectors (tensors) of values, which are on their own context-free data structures. The separation of concerns between the context- and processing-providing models and context-free embeddings allows transparent caching and processing optimizations, where at least part of the cost can be amortized.

3.1. Vector Embeddings with E-Scan

Embeddings represent an intermediate data representation that is contextualized or processed by a corresponding model. They are uniform, no matter the data type,

Table 2
Model inference costs (openai.com/pricing)

Format	Model	Unit	Price ($)
Text	Ada v2	1K tokens	0.0001
Image	Dall-E (1024x1024)	1 image	0.02
Audio	Whisper	1 minute	0.006

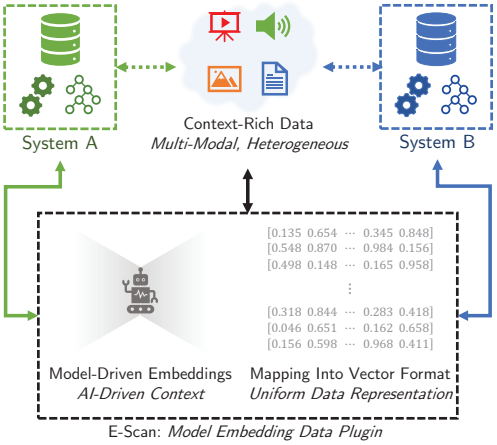

Figure 2: E-Scan: encapsulating model description, original objects, and their embeddings in a plugin/connector design.

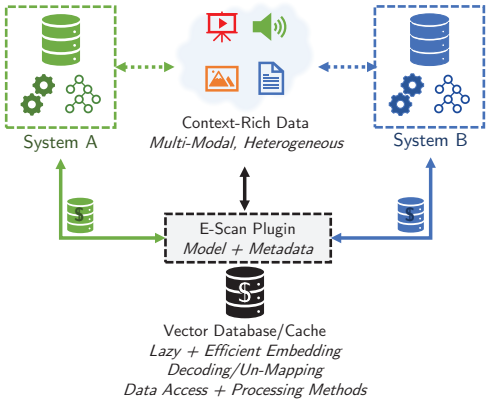

Figure 3: E-Scan data exchange plugin/connector system.

where their dimensionality may vary. This common representation motivates having a vector data management layer that would serve as a caching layer, offering exact and approximate retrieval and data access methods.

Figure 2 shows a conceptual flow of information: rather than System A, System B, and the base data stored in various object stores being exposed as raw sources, we propose a scan connector/plugin-based mechanism called E-Scan. This allows a composable and common plugin for exchanging contextual data. First, the model information is preserved to provide context to the embeddings, along with those transferred and exchanged as vector data. In case the original data is required, and for caching purposes, the object ID is also preserved to provide a link with the original data (black arrow). Then, if System A and System B need to process or exchange context-rich information, the plugin contains all the information and specification of the model, data, and original input in a uniform way.

3.2. Schema and Metadata

Rather than exchanging data and embeddings directly, in an ad-hoc manner, and manually keeping information about the models, we propose storing this provenance information in the plugin metadata. To our best knowledge, there isn't a uniform way for such specifications, but keeping the information about the particular model, creator, origin, and input and output parameters represents minimal information required for exchanging information between different components. Keeping and transferring only the data of need allows lightweight transfer by exchanging only the requested data, and ideally not transferring the original data, rather than only keeping a reference to the original file containing the exact source and identifying key.

Input requirements should indicate the expected data format or a schema and particular characteristics of a type that can be instantiated, such as image size in pixels. The output should equally specify what is expected, an output schema, and the fields and types of the given schema. This can allow model-operator composability in a more traditional relational optimization case [1] or provide a blueprint for generating a code for data exchange or ingestion between different components.

As embedding might not be the first or the last step of models, to enable interoperability with such plugins, the model design needs adaptation to start consuming from a plugin in an intermediate step rather than requiring original input data. Thus, green and blue lines represent the model and data exchange via a common descriptive connector rather than ad-hoc via dashed arrows.

4. E-Scan Plugin and System Design

Data access patterns and the cost of complex model-driven analytics (Section 2) motivated a system design that is pull-based, lazy, and proactively caching expensive operations. This section presents the system aspects of contextual data exchange with E-Scan.

Beyond the specification intended for connecting various components that can customize the schema, to facilitate the data exchange and processing by different components, similar to the ideas behind Apache Arrow [24], insomuch that a lightweight engine and access methods are also part of the design, as presented in Figure 3.

4.1. Caching and Intermediate Storage

The analysis of caching versus recomputing model-based data transformations opens up future work in performance estimation or the benefits of such caching involving data movement of input data and model state that might be larger than working memory, as well as hardware characteristics such as accelerators and interconnects with monetary cost, extending the embedding table caching approaches [25].

The immediate purpose of caching is to avoid expensive model processing. This access path is represented by solid blue and green arrows in Figure 3. As embeddings are all in a common vector format, data management engines specialized for vectors such as Milvus [26] can be used as the caching support layer. Alternatively, lightweight storage and retrieval systems based on vector-based indexes supporting searches on heterogeneous hardware such as FAISS [27] are good candidates for the vector caching and storage layer.

Beyond storing and caching embeddings, basic operations such as similarity or top-K search is often available, allowing more complex data processing and access patterns. Traditional index structures that link the records with their primary keys in original object representations are also necessary to allow fast retrieval, besides full data scans. The object must also be registered to have been embedded by a given model, either explicitly in a data structure or lightweight mechanisms such as bloom filters.

Only in case of a cache miss should the request be propagated to the system and model for explicit embedding (or batched as a group request on the system side). We call this lazy embedding, where this mechanism covers the case of avoiding the expensive path of execution for previously visited objects rather than eagerly embedding all the data. This can happen due to duplicates or re-visiting the same data in different queries. Furthermore, this mechanism can be adapted to avoid embedding if a sufficiently similar entry is already present in embeddings. Still, to approximate similarity in the embedding domain, a cheaper embedding method is required, either through a lightweight model or by applying other input similarity methods that should yield performance improvements, effectively trading off similarity and approximation for computational cost as in traditional approximate query processing.

Finally, as processing might require original objects, not only embeddings, a corresponding key, and the path/identifier is saved to retrieve the object from the corresponding object-store component (black arrow, Figure 3) - as not all use cases and model have a full encoder-decoder architecture that can be used to produce the requested output. This process replaces manual data manipulation and embedding with a generalizable caching mechanism applicable to systems, components, models, or parts of models that create embeddings that can be reused. Naturally, the designation of invalidation or which embeddings might not be good candidates for caching are part of the caching policy.

4.2. Lazy Retrieval and Mapping

The design of E-Scan aims to allow better and more efficient interoperability through a common plugin and interface, to encapsulate functionality and avoid re-implementing desirable data exchange characteristics in complex systems.

In this work, we follow the principles behind NoDB [23], as much as we do not want to embed all the data eagerly. Not all the data might be the object of interest, and this process should be pull-based, meaning that it should happen only upon requesting the embedding. This saves initial processing time and is progressively faster with the previously described caching mechanism in case of cache hits, without any implied prefetching mechanism, just using the access patterns requested by the consumer.

The consumer has to specify to the plugin which model, from which schema, and which data it wishes to transform and provide this information to the connector, for example, by extending industry-led formats [28, 29]. This also involves specifying which objects (context-rich data) should be involved in the query and potentially cached, which can be an explicit list or a result of another query. This mechanism and specification are also similar to adapting to different data types in ViDa [30], where code-generation can also be used to avoid overheads of function calls and create custom-built access patterns and procedures tailored for specific data formats, beyond the common vector representation. This also allows pulling the plugin specification inside system components that support code generation to avoid an explicit component and communication overhead or designing an embedded, process-local component corresponding to what DuckDB is used for in analytical query processing [31].

4.3. Example and Conceptual Use of E-Scan

The main goal of E-Scan is to allow efficient, easy-to-use, and composable interoperability of components that involve embedding and vector data over context-rich, multi-modal formats. In Figure 4, we provide a short example of E-Scan in action. Without this component, the user would be forced to know and implement all the details of caching, processing, optimization, and system interaction, as in Figure 1.

In this example, we use the Segment Anything Model (SAM) [15] as a sample state-of-the-art image processing

tool that segments the objects from the images. We indicate this processing part in green, with corresponding steps under number 1). For text, we use BERT [12] as an example of a word-embedding method that allows semantic similarity matching and classification. Conversely, this processing path is represented in blue, and the corresponding steps are under number 2).

A consumer queries the common interface, starting from the image and text data and instantiated models with corresponding metadata. Suppose this is driven by a query that wants to find all the images with more than three objects which contain cats or dogs, where the corresponding image description (text) is of positive sentiment and contains synonyms of the words "joy" or "cute".

Rather than having this as a manual process, the request comes to the common interface, requesting text and image data processing. This work does not discuss cardinality estimation or cost difference, for example, if first the text embeddings should be executed if cheaper than image embedding. This would consequentially enable a filter pushdown and sideways information passing of more selective processing over the query for finding images that contain more than three objects with cats or dogs, which remains the topic of future work but would be the task of the plugin or corresponding query optimizer as in our recent proposed work of holistic optimization of context-rich engines [1].

In this case, the plugin dispatches the lazy requests to the Image Engine that retrieves the images and performs embedding and processing 1.a). Conversely, image embeddings are created 1.b) and cached along with the original object ID 1.c). If there are similar images based on simpler embeddings or other metrics, the caching mechanism can deploy a similarity search before dispatching requests to the model.

With the existing original image object IDs, we can filter out corresponding text object IDs that qualify and perform the task similarly dispatched to Text Engine lazily. Finally, the requested data is exchanged, and the results of processing and embeddings are exchanged via the common interface for upstream processing.

Some details remain the topic of future work, such as the granularity and the benefit of replacing processing embeddings and intermediate results with data cache. This is because while embedding data is expensive, other post-processing operations might also be useful to cache explicitly. Similarly, this can also motivate modifying existing model architectures and replacing explicit embedding with an intermediate lightweight caching mechanism. Still, the main goal of E-Scan is to provide a blueprint for a common interface for efficient and composable emerging data processing components.

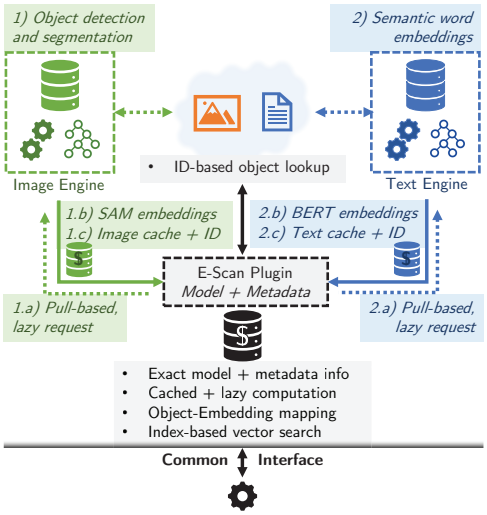

Figure 4: Example system functionalities of E-Scan.

5. Related Work

The provenance of new machine-learning-driven methods creates novel data processing and extraction challenges and opportunities. In our recent work, we propose holistic declarative optimization of context-rich analytical engines meant for hybrid model-relational processing [1].

The challenges of integration of systems and components are related to polystores [4, 5]. In this work, we have tackled the initial blueprint of data exchange primitives and components, still, similar lessons should be applied for cross-system optimization, especially for more complex and hybrid analytical query processing that involves multiple heterogeneous systems.

From the abstractions perspective, this work corresponds to the purpose of Apache Arrow for in-memory formats for flat and hierarchical data [24]. The main idea is to abstract the implementation details and allow easy data exchange between the components, delegating the complexity to the underlying system abstractions handled by the provided metadata and lightweight engine.

Finally, this work relates to NoDB [23] as it does not process nor ingest data before needed, saving on preprocessing cost, considering the cost of processing the entire data which might not be of immediate interest. Instead, we adopt lazy data loading and use caching and appropriate exact and approximate data access patterns to speed up the processing and avoid unnecessary computation, both due to loading and repeated requests to similar data, therefore proposing a new dimension to existing work on embedding table caching [25].

6. Conclusion and Future Work

In this work, we presented E-Scan: the initial vision and proposal for a plugin/connector that intends to simplify and make efficient communication and data exchange between model-driven components in analytical query processing.

We target composability via the common and extensible plugin-based interface that components can invoke, registering the necessary schema information and metadata specific to the particular data and model shape and requirements. Thanks to the common and context-free vector format of embeddings, we propose a reusable and efficient design of caching layer driven by a lightweight vector engine and access methods. This brings closer computationally heavy model processing and trades off part of it for database-inspired caching techniques.

The main goal is to achieve functionality and performance while decoupling the data management details from the intent of the user via lightweight abstractions. Still, there are many available models and use cases that are yet to be tested and remain the topics of future work. Designing a concrete interface with established frameworks and models is required to achieve a well-encompassing set of functionalities and achieve desired ease of use. Finally, exploring the different granularities of replacing embedding computation with caching and standardizing the data exchange protocols is a desired long-term outcome for supporting the holistic integration of model-driven analytics with data management through optimizable, declarative, and composable interfaces and components.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and detailed feedback.

References

[1] V. Sanca, A. Ailamaki, Analytical engines with context-rich processing: Towards efficient next-generation analytics, in: 2023 IEEE 39th International Conference on Data Engineering (ICDE), 2023, pp. 3699–3707. doi:10.1109/ICDE55515.2023.00298.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., {TensorFlow}: a system for {Large-Scale} machine learning, in: 12th USENIX symposium on operating systems design and implementation (OSDI 16), 2016, pp. 265–283.

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

[4] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, S. Zdonik, The bigdawg polystore system, SIGMOD Rec. 44 (2015) 11–16. URL: https://doi.org/10.1145/2814710.2814713. doi:10.1145/2814710.2814713.

[5] M. Karpathiotakis, A. Floratou, F. Özcan, A. Ailamaki, No data left behind: real-time insights from a complex data ecosystem, in: Proceedings of the 2017 Symposium on Cloud Computing, 2017, pp. 108–120.

[6] Amazon Mechanical Turk, Amazon mechanical turk, 2023. URL: https://www.mturk.com/.

[7] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, M. Blum, recaptcha: Human-based character recognition via web security measures, Science 321 (2008) 1465–1468.

[8] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, in: Y. Bengio, Y. LeCun (Eds.), 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013. URL: http://arxiv.org/abs/1301.3781.

[9] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguistics 5 (2017) 135–146. URL: https://doi.org/10.1162/tacl_a_00051. doi:10.1162/tacl_a_00051.

[10] B. Edizel, A. Piktus, P. Bojanowski, R. Ferreira, E. Grave, F. Silvestri, Misspelling oblivious word embeddings, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 3226–3234. URL: https://doi.org/10.18653/v1/n19-1326. doi:10.18653/v1/n19-1326.

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

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

[13] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh,

D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[14] OpenAI, Gpt-4 technical report, 2023. arXiv:2303.08774.

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

[16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 770–778. URL: https://doi.org/10.1109/CVPR.2016.90. doi:10.1109/CVPR.2016.90.

[17] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, I. Sutskever, Zero-shot text-to-image generation, 2021. arXiv:2102.12092.

[18] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, M. D. Plumbley, Panns: Large-scale pretrained audio neural networks for audio pattern recognition, IEEE ACM Trans. Audio Speech Lang. Process. 28 (2020) 2880–2894. URL: https://doi.org/10.1109/TASLP.2020.3030497. doi:10.1109/TASLP.2020.3030497.

[19] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever, Robust speech recognition via large-scale weak supervision, 2022. arXiv:2212.04356.

[20] R. Bommasani, D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, et al., On the opportunities and risks of foundation models, CoRR abs/2108.07258 (2021). URL: https://arxiv.org/abs/2108.07258. arXiv:2108.07258.

[21] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Huggingface's transformers: State-of-the-art natural language processing, arXiv preprint arXiv:1910.03771 (2019).

[22] TensorFlow Hub, 2023. URL: https://www.tensorflow.org/hub.

[23] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, A. Ailamaki, Nodb: efficient query execution on raw data files, in: K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, A. Fuxman (Eds.), Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012, ACM, 2012, pp. 241–252. URL: https://doi.org/10.1145/2213836.2213864. doi:10.1145/2213836.2213864.

[24] Apache Arrow, 2023. URL: https://github.com/apache/arrow.

[25] Z. Wang, Y. Wei, M. Lee, M. Langer, F. Yu, J. Liu, S. Liu, D. G. Abel, X. Guo, J. Dong, J. Shi, K. Li, Merlin hugectr: Gpu-accelerated recommender system training and inference, in: Proceedings of the 16th ACM Conference on Recommender Systems, RecSys '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 534–537. URL: https://doi.org/10.1145/3523227.3547405. doi:10.1145/3523227.3547405.

[26] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, K. Yu, Y. Yuan, Y. Zou, J. Long, Y. Cai, Z. Li, Z. Zhang, Y. Mo, J. Gu, R. Jiang, Y. Wei, C. Xie, Milvus: A purpose-built vector data management system, in: G. Li, Z. Li, S. Idreos, D. Srivastava (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM, 2021, pp. 2614–2627. URL: https://doi.org/10.1145/3448016.3457550. doi:10.1145/3448016.3457550.

[27] J. Johnson, M. Douze, H. Jégou, Billion-scale similarity search with GPUs, IEEE Transactions on Big Data 7 (2019) 535–547.

[28] Understanding AI Plugins in Semantic Kernel and Beyond, 2023. URL: https://learn.microsoft.com/en-gb/semantic-kernel/ai-orchestration/plugins.

[29] OpenAI Platform Plugins, 2023. URL: https://platform.openai.com/docs/plugins/getting-started.

[30] M. Karpathiotakis, I. Alagiannis, T. Heinis, M. Branco, A. Ailamaki, Just-in-time data virtualization: Lightweight data management with vida, in: Seventh Biennial Conference on Innovative Data Systems Research, CIDR 2015, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings, www.cidrdb.org, 2015. URL: http://cidrdb.org/cidr2015/Papers/CIDR15_Paper8.pdf.

[31] M. Raasveldt, H. Mühleisen, Duckdb: an embeddable analytical database, in: Proceedings of the 2019 International Conference on Management of Data, 2019, pp. 1981–1984.