Proof of Concept: Leveraging Pre-trained Models for Merchant Name Matching

Project Overview: The Enhanced Merchant Name Matching Pipeline aims to accurately connect varied merchant names with their canonical forms using a hybrid approach combining semantic understanding with traditional string matching.

Challenge: Effectively utilizing advanced pre-trained language models like BERT and MPNet for this task presents several challenges, especially without the option to fine-tune the model and without deep NLP expertise within the team.

1. Complexity of Training Pre-trained Models:

Training models like BERT and MPNet from scratch requires significant computational resources (e.g., GPUs) and extensive datasets (over 160GB of text for MPNet).[1] This process also demands a deep understanding of machine learning principles, model architectures (like Transformer), and hyperparameter optimization.[3] For most teams without dedicated resources and expertise, this approach is often impractical.[3]

2. Expertise Required for Full Pre-trained Model Utilization:

Working directly with the base BERT or MPNet models from libraries like Hugging Face Transformers involves understanding various components such as tokenizers, model configurations, and different output layers.[1] Extracting meaningful features and implementing custom logic for tasks like semantic similarity requires a strong foundation in NLP and deep learning.[3] The sheer size of these models (e.g., MPNet can be around 6.91GB [User's input]) can also pose challenges in terms of loading, processing, and memory management, requiring specialized coding knowledge.[5]

3. Simpler Alternative: Direct Inference with Sentence Transformers:

Libraries like Sentence Transformers offer a more accessible way to leverage the power of pre-trained models for tasks like semantic similarity.[6] Sentence Transformers builds upon models like MPNet and provides pre-configured models (e.g., all-mpnet-base-v2) that are specifically designed to generate high-quality sentence embeddings.[6] These embeddings are dense vector representations that capture the semantic meaning of text, making it straightforward to calculate the similarity between merchant names.[7]

Example using Sentence Transformers (Conceptual):

Python

```python
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')
merchant_names =
embeddings = model.encode(merchant_names)
# Calculate cosine similarity between embeddings to find matches
```

This approach significantly reduces the complexity, allowing teams without deep NLP expertise to quickly implement a prototype for semantic merchant name matching.[7]

4. Trade-offs and Future Considerations:

While direct inference with Sentence Transformers provides a rapid prototyping solution, it might have limitations compared to fine-tuning a model on a domain-specific dataset of merchant names. Fine-tuning allows the model to learn the nuances and specific patterns relevant to merchant data, potentially leading to higher accuracy.[3] However, fine-tuning requires labeled data and computational resources, which are currently unavailable.

Conclusion:

Given the current constraints of not being able to train a model and the limited NLP expertise within the team, utilizing a library like Sentence Transformers for direct inference with a pre-trained model like all-mpnet-base-v2 offers a practical and efficient way to demonstrate a proof of concept for the Enhanced Merchant Name Matching Pipeline.[7] This approach allows us to leverage the semantic understanding capabilities of advanced models without the complexities of training or in-depth model manipulation. While other more advanced options exist, they would necessitate significant NLP coding expertise and time investment.[3] This simpler method provides a valuable starting point for showcasing the potential of semantic matching in the pipeline.

References:

- [1] MPNet documentation in Hugging Face Transformers: https://github.com/huggingface/transformers/blob/main/docs/source/en/model_doc/mpnet.md
- [6] Overview of All-MPNet-Base-V2:

https://zilliz.com/learn/all-mpnet-base-v2-enhancing-sentence-embedding-with-ai
- [3] A Complete Guide to BERT with Code: https://towardsdatascience.com/a-complete-guide-to-bert-with-code-9f87602e4a11/
- [9] All-MPNet-Base-V2 model card in Sentence Transformers: https://huggingface.co/sentence-transformers/all-mpnet-base-v2[1]
- [7] Dataloop overview of Sentence Transformers All Mpnet Base V2: https://dataloop.ai/library/model/sentence-transformers_all-mpnet-base-v2/
- [3] A Complete Guide to BERT with Code (Part 4): https://towardsdatascience.com/a-complete-guide-to-bert-with-code-9f87602e4a11/
- [8] Paraphrase-MPNet-Base-V2 model card in Sentence Transformers: https://huggingface.co/sentence-transformers/paraphrase-mpnet-base-v2
- [7] Dataloop overview of Sentence Transformers All Mpnet Base V2: https://dataloop.ai/library/model/sentence-transformers_all-mpnet-base-v2/
- [2] MPNet: Masked and Permuted Pre-training for Language Understanding:(https://www.microsoft.com/en-us/research/uploads/prod/2020/11/NIPS_MPNet.pdf)
- [11] MPNet base trained on Natural Questions pairs: https://huggingface.co/tomaarsen/mpnet-base-nq-prompts
- [5] Feature request for MPNet in text-embeddings-inference: https://github.com/huggingface/text-embeddings-inference/issues/33
- [1] MPNet documentation in Hugging Face Transformers: https://github.com/huggingface/transformers/blob/main/docs/source/en/model_doc/mpnet.md
- [2] MPNet Model Architecture and Pre-training:(https://www.microsoft.com/en-us/research/uploads/prod/2020/11/NIPS_MPNet.pdf)
- [10] Pretrained Models in Sentence Transformers: https://www.sbert.net/docs/sentence_transformer/pretrained_models.html
- [4] MPNet usage in Hugging Face Transformers: https://huggingface.co/docs/transformers/en/model_doc/mpnet

Sources

1. https://github.com/jesterlabs/vexpresso
2. https://github.com/vespaai/cloud

Utilizing Pre-trained Language Models: An Evaluation of Common Approaches

Pre-trained language models (PLMs) such as BERT and MPNet have revolutionized the field of Natural Language Processing (NLP) by offering robust and adaptable language representations.[1] These models, pre-trained on vast amounts of text data, acquire a broad understanding of language semantics and syntax, which can then be leveraged for a multitude of downstream tasks.[11] BERT, introduced by Google AI Language, stands as a landmark achievement in this domain, demonstrating significant advancements across various NLP benchmarks.[1] Building upon the success of models like BERT, MPNet introduced an innovative pre-training methodology by combining the strengths of Masked Language Modeling (MLM) and Permuted Language Modeling (PLM), effectively addressing some of the inherent limitations of its predecessors.[2] Consequently, both BERT and MPNet have found widespread application in diverse NLP tasks, including semantic search, text classification, and question answering.[3]

The user has presented a statement outlining three potential options for utilizing these pre-trained models and seeks to verify the accuracy of this statement. The proposed options are: training the model, using a fully functional model (requiring high expertise), and using a hardcoded version (less reliable). This report aims to analyze the validity of these options, explore alternative strategies for leveraging PLMs, discuss the expertise required for each approach, and provide a comprehensive understanding of the landscape of pre-trained model utilization.

Understanding the Landscape of Pre-trained Model Utilization

2.1. Training Pre-trained Models from Scratch

Training a pre-trained language model from the ground up is an undertaking that demands substantial computational resources and a deep understanding of machine learning principles.[13] For models like MPNet, the pre-training process involves training on datasets exceeding 160GB of text.[2] This process often necessitates the use of specialized hardware infrastructure, such as the 32 NVIDIA Tesla V100 GPUs utilized during the pre-training of MPNet.[4] Furthermore, it requires careful selection of the model architecture, such as the Transformer architecture employed by both BERT and MPNet, which is specifically designed to process sequential data effectively.[1] Defining appropriate pre-training objectives is also critical; BERT utilizes Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [1], while MPNet employs its unique masked and permuted language modeling approach.[2] Optimizing the numerous hyperparameters involved in training these complex models adds another layer of difficulty. The financial implications of such endeavors are also significant, with the training of state-of-the-art models like GPT-4 reportedly costing upwards of $100 million.[23] Consequently, training PLMs from scratch is an option primarily pursued by large research institutions and corporations that possess the necessary resources, infrastructure, and specialized expertise in deep learning.[13] For the vast majority of individual developers and organizations, this approach is often impractical due to the prohibitive costs and the level of expertise required.

2.2. Fine-tuning Pre-trained Models

A more common and effective strategy for leveraging the power of pre-trained language models is fine-tuning.[1] Fine-tuning involves taking a pre-trained model and further training it on a smaller, task-specific labeled dataset.[1] The key to successful fine-tuning lies in selecting a pre-trained model that is relevant to the intended downstream application.[19] Often, this process involves adding a task-specific layer on top of the pre-trained model's architecture. For instance, in sentiment analysis, a classification head with a specific number of output neurons corresponding to the sentiment classes can be added.[1] Fine-tuning offers several advantages over training from scratch, including significantly reduced training time, lower computational costs, and the need for less task-specific data.[1] Various techniques exist for fine-tuning PLMs, ranging from updating all the model's parameters (full fine-tuning) to more parameter-efficient methods like Low-Rank Adaptation (LoRA) and adapter tuning.[28] These parameter-efficient techniques are particularly popular for fine-tuning very large language models as they require fewer computational resources. The fine-tuning process typically involves preparing the task-specific dataset, tokenizing the text data using the pre-trained model's tokenizer, and then setting up a training loop using machine learning libraries like Hugging Face Transformers.[1] As an example, MPNet has been successfully fine-tuned for various tasks, including semantic text similarity using contrastive learning objectives [35] and for achieving strong performance on the General Language Understanding Evaluation (GLUE) benchmark.[4] Fine-tuning provides a practical and efficient way to adapt the general language understanding capabilities of pre-trained models to specific application requirements, demanding a moderate level of expertise in machine learning and NLP.

2.3. Direct Inference with Pre-trained Models

Pre-trained language models can also be utilized directly for inference on a wide range of NLP tasks without requiring any additional training.[19] This approach involves loading a pre-trained model and its associated tokenizer and then feeding input data to the model to obtain outputs such as predictions or embeddings.[19] Libraries like Hugging Face Transformers have significantly simplified this process by providing user-friendly functions like pipeline(), which abstract away much of the underlying complexity.[38] For example, MPNet models, such as All-MPNet-Base-V2, are specifically designed for direct use in tasks like semantic search, clustering of text, and measuring the similarity between sentences.[19] These models excel at generating sentence embeddings, which are dense vector representations of text that capture the semantic meaning of sentences, enabling efficient comparison and retrieval of semantically similar text.[19] Furthermore, pre-trained models can be used for zero-shot classification, where the model can categorize text according to a set of provided labels, even if it has not been explicitly trained on those specific labels.[38] Direct inference provides an accessible means to harness the power of PLMs for various NLP applications, often requiring a relatively lower level of expertise compared to training or even fine-tuning, especially when utilizing high-level libraries.

2.4. Addressing the Concept of "Hardcoded Versions"

The term "hardcoded versions" in the context of pre-trained models can be interpreted in several ways. One possibility is that it refers to using a specific, unchangeable release of a pre-trained model without incorporating any subsequent updates or engaging in fine-tuning.[8] Another interpretation might involve the use of simplified or distilled versions of a model, where the model's size and complexity are reduced to improve efficiency for specific use cases.[54] Techniques such as knowledge distillation and model compression, including quantization and pruning, are employed to create these smaller, more efficient models.[54] A third, and perhaps most literal, interpretation could be an attempt to manually implement certain aspects of a pre-trained model's functionality directly in code. Regarding reliability, using a fixed version of a pre-trained model might lead to the model becoming outdated as newer, more improved versions are released.[8] Simplified models, while offering efficiency benefits, might potentially sacrifice some accuracy or generalizability compared to their larger counterparts.[54] Most significantly, attempting to manually recreate the intricate functionality of a pre-trained model from scratch would almost certainly result in a system that lacks the robustness, accuracy, and performance of the original, highly optimized models, given the complexity of these architectures and the extensive training they undergo. It is also worth noting a different context where "hardcoded" is relevant: the security risk associated with embedding sensitive information, such as API keys, directly within code related to model usage.[55] However, this is distinct from the notion of a "hardcoded version" of the model itself. In conclusion, while using a specific version of a pre-trained model is standard practice, and simplified versions exist, the idea of a generally unreliable "hardcoded version" as a primary option for utilizing PLMs is inaccurate and potentially misleading.

3. Evaluating the Accuracy of the User's Statement

Based on the analysis above, the user's statement presents a partially correct but oversimplified view of the options for utilizing pre-trained language models. The first option, **training**, is indeed a possibility, but as discussed, it is often impractical for most users due to the significant resource and expertise requirements. The second option, using a **fully functional model**, which likely refers to direct inference, is also a valid approach. However, the assertion that this always requires **high expertise** is not entirely accurate. While a foundational understanding of NLP concepts is beneficial, the advent of user-friendly libraries like Hugging Face Transformers has made direct inference accessible to individuals with a relatively lower level of expertise for many common tasks. The third option, using a **hardcoded version**, which is described as **less reliable**, is the most problematic. Manually implementing the functionality of a pre-trained model is an exceptionally difficult and likely unreliable endeavor. While using a specific version of a model has its considerations, and simplified models exist, neither accurately fits the description of a generally unreliable "hardcoded version" as implied by the user. Furthermore, a significant omission from the user's statement is the crucial method of **fine-tuning**. As detailed in section 2.2, fine-tuning is a widely adopted and highly effective strategy for adapting pre-trained models to specific downstream tasks, offering a balance between leveraging the model's pre-existing knowledge and tailoring it to particular application needs. The level of expertise required for each of these methods varies considerably, further highlighting the oversimplification in the user's statement.

4. Exploring Alternative Approaches to Leverage Pre-trained Models

Beyond the three options presented by the user, several other important approaches exist for leveraging the capabilities of pre-trained language models.

4.1. Prompt Engineering and In-Context Learning

Prompt engineering is a technique that focuses on designing effective input prompts to guide a pre-trained language model towards generating the desired outputs without the need for explicitly fine-tuning the model's parameters.[23] This involves carefully crafting the input text to provide sufficient context, clearly specify the desired output format, and sometimes include delimiters to help the model focus on the relevant information.[23] In-context learning is a related paradigm where the language model adapts its responses based on the specific queries or prompts it receives.[30] This can involve zero-shot prompting, where no examples are provided, or few-shot prompting, where a small number of input-output examples are included in the prompt to demonstrate the desired behavior.[30] These techniques offer efficient ways to interact with PLMs for specific tasks, often requiring a moderate level of expertise focused on understanding how to best communicate with the model through well-designed prompts rather than deep technical knowledge of the model's internal workings.

4.2. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an approach that aims to enhance the accuracy and relevance of language model responses by integrating information retrieved from external knowledge sources.[24] When a user poses a query, the RAG system first retrieves relevant documents or information from a database or knowledge base based on the query. This retrieved information is then combined with the original query and fed into the language model to generate a more informed and contextually accurate response.[24] RAG is particularly beneficial for tasks that require access to up-to-date information or domain-specific knowledge that the pre-trained model might not inherently possess, providing a way to improve performance without the need for extensive fine-tuning of the model itself. Implementing RAG typically requires a moderate level of expertise, involving an understanding of information retrieval techniques and how to effectively integrate external data with a language model.

4.3. Utilizing Pre-trained Models as Feature Extractors

Pre-trained language models can also be employed as powerful feature extractors.[13] In this approach, the intermediate layer activations of the pre-trained model, rather than the final prediction layer, are used as meaningful features representing the input data.[13] These extracted features can then be used as input to train other, often simpler, machine learning models for specific downstream tasks. For example, in image classification, features extracted from the convolutional layers of a pre-trained Convolutional Neural Network (CNN) can be used to train a Support Vector Machine (SVM) classifier.[59] This technique, known as transfer learning, can be computationally more efficient and might require less labeled data compared to fine-tuning the entire pre-trained model or training a model from scratch.[59] Using pre-trained models for feature extraction demands a moderate understanding of feature extraction concepts and the ability to access and utilize the intermediate representations of these models.

5. Expertise Requirements for Different Utilization Methods

The level of expertise required to effectively utilize pre-trained language models varies significantly depending on the chosen method:

Method	Expertise Required	Reliability	Resource Intensity
Training from Scratch	High	Potentially high, but requires significant effort to achieve and validate	Very High
Fine-tuning	Moderate to High	High, especially when using appropriate data and techniques	Moderate
Direct Inference	Low to Moderate	Generally high for well-established models	Low
Prompt Engineering & In-Context Learning	Moderate	Varies depending on the complexity of the task and the effectiveness of prompts	Very Low
Retrieval-Augmented Generation (RAG)	Moderate	High, especially for knowledge-intensive tasks	Low to Moderate
Utilizing Pre-trained Models as Feature Extractors	Moderate	Good, depends on the pre-trained model and the task-specific model trained on top	Low

Training a pre-trained model from scratch demands a profound understanding of deep learning principles, model architectures, training methodologies, hyperparameter tuning, and the management of significant computational resources, necessitating a high level of expertise.[13] Fine-tuning requires a solid foundation in machine learning and NLP, familiarity with pre-trained model architectures, proficiency in data handling and tokenization, and understanding of training procedures and evaluation metrics; the level of expertise can range from moderate to high.[1] For basic tasks, direct inference using high-level libraries like Hugging Face Transformers can be achieved with a relatively low level of expertise, primarily requiring an understanding of the model's intended use and the library's API; however, more complex scenarios or debugging might necessitate a deeper understanding, placing it at a low to moderate level.[22] Prompt engineering and in-context learning demand a good intuition for how language models respond to different prompts and the ability to design prompts that effectively guide the model's output, requiring a moderate level of expertise focused on model behavior and creative prompt design.[23] Implementing RAG requires a moderate level of expertise, involving knowledge of information retrieval techniques and the integration of external data with language models.[24] Finally, using pre-trained models for feature extraction also requires a moderate understanding of feature extraction concepts and the ability to access and utilize the intermediate layers of these models.[59]

6. Conclusion

In conclusion, the user's statement regarding the three options for using pre-trained models is partially accurate but oversimplified and contains some inaccuracies. While training from scratch and direct inference are indeed possibilities, the former is often impractical for most users, and the latter does not always necessitate high expertise, especially with the aid of modern libraries. The concept of a generally unreliable "hardcoded version" is not a standard or recommended approach. Furthermore, the user's statement omits the crucial and widely used method of fine-tuning. The primary ways to leverage pre-trained language models include the resource-intensive option of training from scratch (typically for large organizations), the common and effective method of fine-tuning for task-specific adaptation, the increasingly accessible approach of direct inference, and other valuable techniques such as prompt engineering, Retrieval-Augmented Generation (RAG), and utilizing models as feature extractors. The choice of method depends on the specific task, available resources, required expertise, and desired level of performance. For most users, fine-tuning or direct inference offer more practical starting points than training a model from scratch, and the reliability of a "hardcoded version" is questionable. It is highly recommended to explore the resources and tools provided by libraries like Hugging Face Transformers [38], which significantly simplify the implementation and access to a vast array of pre-trained models.

Works cited

1. A Complete Guide to BERT with Code | Towards Data Science, accessed April 17,

2025,
https://towardsdatascience.com/a-complete-guide-to-bert-with-code-9f87602e4a11/
2. transformers/docs/source/en/model_doc/mpnet.md at main ... - GitHub, accessed April 17, 2025,
https://github.com/huggingface/transformers/blob/main/docs/source/en/model_doc/mpnet.md
3. Bringing Visibility to Workflow Processes via Topic Modeling with BERT Transformer Models, accessed April 17, 2025,
https://erepo.uef.fi/bitstream/handle/123456789/28478/urn_nbn_fi_uef-20221236.pdf?sequence=1&isAllowed=y
4. www.microsoft.com, accessed April 17, 2025,
https://www.microsoft.com/en-us/research/uploads/prod/2020/11/NIPS_MPNet.pdf
5. MPNet: Masked and Permuted Pre-training for Language Understanding - arXiv, accessed April 17, 2025, https://arxiv.org/abs/2004.09297
6. All-Mpnet-Base-V2: Enhancing Sentence Embedding with AI - Zilliz Learn, accessed April 17, 2025,
https://zilliz.com/learn/all-mpnet-base-v2-enhancing-sentence-embedding-with-ai
7. Towards Sentence Level Inference Attack Against Pre-trained Language Models - Privacy Enhancing Technologies Symposium, accessed April 17, 2025,
https://petsymposium.org/popets/2023/popets-2023-0070.pdf
8. Exploring the Lifecycle and Maintenance Practices of Pre-Trained Models in Open-Source Software Repositories - arXiv, accessed April 17, 2025,
https://arxiv.org/html/2504.06040v1
9. Learning Sample Difficulty from Pre-trained Models for Reliable Prediction, accessed April 17, 2025,
https://proceedings.neurips.cc/paper_files/paper/2023/file/50251f54848a433f3e47ae3b7cbded53-Paper-Conference.pdf
10. A Primer in BERTology: What We Know About How BERT Works | Transactions of the Association for Computational Linguistics - MIT Press Direct, accessed April 17, 2025,
https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00349/96482/A-Primer-in-BERTology-What-We-Know-About-How-BERT
11. Explanation of Pre-Trained Model | Sapien's AI Glossary, accessed April 17, 2025,
https://www.sapien.io/glossary/definition/pre-trained-model
12. Pre Trained Model Definition | Encord, accessed April 17, 2025,
https://encord.com/glossary/pre-trained-model-definition/
13. Understanding pre-trained AI models and their applications - Nebius, accessed April 17, 2025,
https://nebius.com/blog/posts/understanding-pre-trained-ai-models
14. Pre Trained Models - Lark, accessed April 17, 2025,
https://www.larksuite.com/en_us/topics/ai-glossary/pre-trained-models
15. What Is a Pretrained AI Model? - NVIDIA Blog, accessed April 17, 2025,

https://blogs.nvidia.com/blog/what-is-a-pretrained-ai-model/
16. AI/ML Model and Pre-Trained Weight Packaging in Fedora - legal, accessed April 17, 2025, https://lists.fedoraproject.org/archives/list/legal@lists.fedoraproject.org/thread/PIPILJCMDEO67ORL4SAKB3NPHHVMFDJE/
17. Top 5 PreTrained Models in Natural Language Processing (NLP) | GeeksforGeeks, accessed April 17, 2025, https://www.geeksforgeeks.org/top-5-pre-trained-models-in-natural-language-processing-nlp/
18. MPNet - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/en/model_doc/mpnet
19. All Mpnet Base V2 · Models - Dataloop, accessed April 17, 2025, https://dataloop.ai/library/model/sentence-transformers_all-mpnet-base-v2/
20. Paraphrase Mpnet Base V2 · Models - Dataloop, accessed April 17, 2025, https://dataloop.ai/library/model/sentence-transformers_paraphrase-mpnet-base-v2/
21. sentence-transformers/all-mpnet-base-v2 - Hugging Face, accessed April 17, 2025, https://huggingface.co/sentence-transformers/all-mpnet-base-v2
22. Why You Should Use Pre-Trained Models Versus Building Your Own - Cohere, accessed April 17, 2025, https://cohere.com/blog/pre-trained-vs-in-house-nlp-models
23. Deploying Large Language Models - Everything to Consider - MonsterAPI Blog, accessed April 17, 2025, https://blog.monsterapi.ai/llm-deployment-best-practices/
24. Successful LLM Deployment in 5 steps: Strategies & Best Practices - MidShift Blog, accessed April 17, 2025, https://blog.midshift.co.uk/career-development/successful-llm-deployment-in-5-steps-strategies-best-practices/
25. Train and Fine-Tune Sentence Transformer Models - Marqo, accessed April 17, 2025, https://www.marqo.ai/course/training-fine-tuning-sentence-transformers
26. Training Overview — Sentence Transformers documentation, accessed April 17, 2025, https://sbert.net/docs/sentence_transformer/training_overview.html
27. [2306.10790] Preserving Commonsense Knowledge from Pre-trained Language Models via Causal Inference - arXiv, accessed April 17, 2025, https://arxiv.org/abs/2306.10790
28. Prompt Engineering vs. Fine-Tuning—Key Considerations and Best Practices | Nexla, accessed April 17, 2025, https://nexla.com/ai-infrastructure/prompt-engineering-vs-fine-tuning/
29. Pre-Training vs Fine Tuning: Choosing the Right Approach - Label Your Data, accessed April 17, 2025, https://labelyourdata.com/articles/llm-fine-tuning/pre-training-vs-fine-tuning
30. Fine-tuning large language models (LLMs) in 2025 - SuperAnnotate, accessed April 17, 2025, https://www.superannotate.com/blog/llm-fine-tuning
31. The Ultimate Guide to Building Large Language Models - Multimodal.dev, accessed April 17, 2025,

https://www.multimodal.dev/post/the-ultimate-guide-to-building-large-language-models
32. How [And Why] To Train Your Own Language Model - Coralogix, accessed April 17, 2025, https://coralogix.com/ai-blog/how-to-train-your-own-language-model/
33. SLM vs LoRA LLM: Edge Deployment and Fine-Tuning Compared - Prem, accessed April 17, 2025, https://blog.premai.io/slm-vs-lora-llm-edge-deployment-and-fine-tuning-compared/
34. Comparing LLM fine-tuning methods - SignalFire, accessed April 17, 2025, https://www.signalfire.com/blog/comparing-llm-fine-tuning-methods
35. Finetune Sentence Transformer using Huggingface Trainer API - Stack Overflow, accessed April 17, 2025, https://stackoverflow.com/questions/77842257/finetune-sentence-transformer-using-huggingface-trainer-api
36. Fine-tuning MPNet on GLUE tasks - GitHub, accessed April 17, 2025, https://github.com/microsoft/MPNet/blob/master/MPNet/README.glue.md
37. Using Pre-trained Language Model — gluonnlp 0.10.0 documentation, accessed April 17, 2025, https://nlp.gluon.ai/examples/language_model/use_pretrained_lm.html
38. Transformers, what can they do? - Hugging Face LLM Course, accessed April 17, 2025, https://huggingface.co/learn/llm-course/chapter1/3
39. What are Hugging Face Transformers? - Databricks Documentation, accessed April 17, 2025, https://docs.databricks.com/aws/en/machine-learning/train-model/huggingface/
40. OpenSearch-provided pretrained models, accessed April 17, 2025, https://opensearch.org/docs/latest/ml-commons-plugin/pretrained-models/
41. Pretrained Models — Sentence Transformers documentation, accessed April 17, 2025, https://www.sbert.net/docs/sentence_transformer/pretrained_models.html
42. Optimum pipelines for inference - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/optimum/v1.2.3/pipelines
43. Pipelines for inference - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/v4.45.2/pipeline_tutorial
44. Pipeline - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/pipeline_tutorial
45. Pipelines for inference - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/v4.27.1/pipeline_tutorial
46. Pipelines for inference - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/v4.24.0/pipeline_tutorial
47. HuggingFace Pipeline Use Cases - YouTube, accessed April 17, 2025, https://www.youtube.com/watch?v=B_gH99hXbZQ
48. Transformers - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/index
49. Hugging Face Transformers: Leverage Open-Source AI in Python, accessed April 17, 2025, https://realpython.com/huggingface-transformers/
50. What Transformers can do - Hugging Face, accessed April 17, 2025,

https://huggingface.co/docs/transformers/task_summary
51. Quickstart - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/quicktour
52. Quick tour - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers//quicktour
53. Pipelines for inference - Hugging Face, accessed April 17, 2025, https://huggingface.co/docs/transformers/v4.17.0/en/pipeline_tutorial
54. Efficient AI in Practice: Training and Deployment of Efficient LLMs for Industry Applications - arXiv, accessed April 17, 2025, https://arxiv.org/html/2502.14305v1
55. Yes, GitHub's Copilot can Leak (Real) Secrets - GitGuardian Blog, accessed April 17, 2025, https://blog.gitguardian.com/yes-github-copilot-can-leak-secrets/
56. Early Detection of Hard-Coded Secrets in Software Development: A Multi-Method Approach Integrating Static Analysis, Entropy-Based Detection, and Machine Learning - ResearchGate, accessed April 17, 2025, https://www.researchgate.net/publication/386557724_Early_Detection_of_Hard-Coded_Secrets_in_Software_Development_A_Multi-Method_Approach_Integrating_Static_Analysis_Entropy-Based_Detection_and_Machine_Learning
57. Pre-training vs Fine-Tuning vs In-Context Learning of Large Language Models, accessed April 17, 2025, https://www.entrypointai.com/blog/pre-training-vs-fine-tuning-vs-in-context-learning-of-large-language-models/
58. What is the difference between pre-training, fine-tuning, and instruct-tuning exactly? - Reddit, accessed April 17, 2025, https://www.reddit.com/r/learnmachinelearning/comments/19f04y3/what_is_the_difference_between_pretraining/
59. Extract Features from Image using Pretrained Model | Deep Learning | Python, accessed April 17, 2025, https://www.hackersrealm.net/post/extract-features-from-image-python
60. Can using a pre-trained deep learning model as the feature extractor in the bag-of-deep-visual-words model always improve image classification accuracy? - PubMed, accessed April 17, 2025, https://pubmed.ncbi.nlm.nih.gov/38422007/
61. Extract Image Features Using Pretrained Network - MathWorks, accessed April 17, 2025, https://www.mathworks.com/help/deeplearning/ug/extract-image-features-using-pretrained-network.html
62. Autoencoder vs Pre-trained network for feature extraction - Data Science Stack Exchange, accessed April 17, 2025, https://datascience.stackexchange.com/questions/113942/autoencoder-vs-pre-trained-network-for-feature-extraction
63. What's the purpose of "feature extraction using a pretrained model"? - AI Stack Exchange, accessed April 17, 2025, https://ai.stackexchange.com/questions/40269/whats-the-purpose-of-feature-extraction-using-a-pretrained-model
64. An evaluation of pre-trained models for feature extraction in image classification - arXiv, accessed April 17, 2025, https://arxiv.org/abs/2310.02037

65. From English to foreign languages: transferring pre-trained language models - Amazon Science, accessed April 17, 2025, https://www.amazon.science/publications/from-english-to-foreign-languages-transferring-pre-trained-language-models