Definição de Acordos de Nível de Serviço Personalizados para Treinamento de Modelos via Consultas *k-Skyband*

Gustavo Santos, Marcos Bedo, Yuri Frota, Daniel de Oliveira

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)

gassantos@id.uff.br,{marcosbedo,yuri,danielcmo}@ic.uff.br

Abstract. *Training Machine Learning (ML) models requires configuring hyperparameters and the execution environment, leading to varying execution times and energy consumption. In the cloud, users must define Service Level Agreements (SLAs) and select computational resources based on the characteristics of the training process. In this paper, we extend the concept of Personalized Service Level Agreement (PSLA) by proposing* `PSLA4ML`*, which enables specifying requirements and retrieving resource configurations and their costs for executing the model training workflow.* `PSLA4ML`*s are computed as k-Skyband queries, considering multiple metrics, e.g., execution time, financial cost, etc. The approach was evaluated using BERT-family model training workflows, yielding promising results.*

Resumo. *O treinamento de modelos de Aprendizado de Máquina (ML) envolve a configuração de parâmetros, hiperparâmetros e do ambiente de execução, o que resulta em tempos de execução e consumo de energia distintos. Em ambientes de nuvem, os usuários precisam definir Acordos de Nível de Serviço (SLA), escolhendo recursos computacionais com base nas características do treinamento. Neste artigo, estendemos o conceito de Personalized Service Level Agreement (PSLA), propondo o* `PSLA4ML`*, que permite especificar requisitos e obter configurações de recursos, com custos e métricas associados ao workflow de treinamento do modelo. Os* `PSLA4ML` *são gerados por meio de consultas k-Skyband, considerando múltiplas métricas, como tempo, custo financeiro , etc. A abordagem foi avaliada por meio de workflows de treinamento de modelos da família BERT, o que indica sua viabilidade.*

1. Introdução

Na última década, o uso de modelos de Aprendizado de Máquina como parte do processo de decisão tornou-se um padrão *de facto* em diversas aplicações [Lu et al. 2023]. Esse interesse intensificou-se com a publicação da AlexNet [Krizhevsky et al. 2017], que impulsionou os avanços em Aprendizado Profundo (DL) e possibilitou melhorias no desempenho de modelos em tarefas complexas. Esse movimento aumentou com o surgimento de modelos de linguagem, *e.g.*, BERT [Devlin et al. 2019], e com os Modelos de Linguagem de Larga Escala (LLMs) [Zhao et al. 2023], que contribuíram para uma nova onda de aplicações. Modelos de ML tipicamente requerem um *workflow* de treinamento composto por várias atividades [Amorim et al. 2026]. Nesse *workflow*, um conjunto de dados é utilizado para treinar um modelo capaz de inferir sobre novos dados. Embora esse princípio seja comum a diferentes técnicas, o custo de treinamento varia conforme os dados e a complexidade do modelo. Por exemplo, no treinamento de um modelo de árvores de decisão, encontrar regiões que separem as classes no espaço de características requer poucas passadas sobre os dados, enquanto o treinamento de LLMs envolve a aprendizagem interativa de padrões linguísticos complexos a partir de grandes volumes de dados e de bilhões de parâmetros [Zhao et al. 2023].

Diante desse cenário, o uso de ambientes de Computação de Alto Desempenho (HPC) tornou-se indispensável para o treinamento de LLMs. No entanto, muitos pesquisadores não dispõem da infraestrutura necessária, *e.g.*, GPUs e *clusters* dedicados, devido aos custos de aquisição e manutenção. Como alternativa, é comum recorrer a provedores de nuvem que oferecem recursos sob demanda. Apesar dessa flexibilidade, a variedade de recursos disponíveis torna a definição de Acordos de Nível de Serviço (SLA) uma tarefa complexa. Em geral, é necessário decidir tanto a quantidade quanto os tipos de recursos a serem instanciados, considerando fatores como o número de CPUs/GPUs e a memória disponível. Essa decisão exige conhecimento técnico especializado e a complexidade do ambiente pode levar os usuários a superestimar ou subestimar os recursos necessários. Um problema semelhante ocorre no processamento de consultas de SGBDs na nuvem, *e.g.*, MyriaDB [Halperin et al. 2014], em que os usuários também enfrentam dificuldades para definir SLAs adequados às suas consultas. Nesse contexto, foi proposto o conceito de *Personalized Service Level Agreement* (PSLA) [Ortiz et al. 2015], que permite ao usuário fornecer informações sobre o esquema e as estatísticas básicas de seus dados, bem como sobre o ambiente de nuvem a ser utilizado.

A partir dessas informações, a abordagem pode estimar quais tipos de consulta podem ser executados e qual desempenho pode ser esperado em diferentes níveis de serviço, cada um associado a um custo específico. Assim, o PSLA busca simplificar o processo de configuração de recursos ao abstrair parte da complexidade do ambiente de nuvem. Entretanto, o conceito de PSLA não se aplica diretamente ao cenário de treinamento de modelos de ML. Isso ocorre porque ele foi concebido para consultas algébricas/SQL, cujas características diferem das observadas em modelos de ML. Além disso, o PSLA considera apenas o tempo de execução e o custo financeiro como métricas principais, desconsiderando outros aspectos relevantes, *e.g.*, consumo energético e emissão de carbono, que podem ser importantes para diferentes *stakeholders* [Talaei Khoei et al. 2023].

Com o objetivo de suprir essa lacuna, este artigo propõe uma extensão do conceito de PSLA voltada ao treinamento de modelos de ML, denominada `PSLA4ML` (*Personalized Service Level Agreement for Machine Learning*). A proposta consiste em definir diferentes níveis de serviço (*i.e.*, *tiers*) que sirvam como modelos de referência para a submissão de *workflows* de treinamento com estimativas prévias de métricas relevantes, como custo de execução, tempo de treinamento, consumo energético, etc. Dessa forma, os usuários passam a ter uma visão das implicações de cada configuração antes de iniciar o treinamento de seus modelos na nuvem. Os *tiers* propostos são definidos com base nos resultados de consultas do tipo *k-Skyband* [Ciaccia and Martinenghi 2020, Ciaccia and Martinenghi 2025] sobre um banco de dados de proveniência que armazena informações históricas de execuções, *i.e.*, *traces*, de *workflows*. A abordagem foi avaliada por meio da execução de um workflow de treinamento de modelos da família BERT em diferentes configurações de recursos computacionais. Os resultados indicam que a proposta é viável e eficaz para apoiar a definição de `PSLA4ML`.

2. Conceitos e Trabalhos Relacionados

PSLA. Um *Personalized Service Level Agreement* (PSLA) pode ser definido como um conjunto de *tiers* $PSLA = \{R_1, R_2, \ldots, R_k\}$, no qual cada *tier* R_i representa um nível de serviço distinto, caracterizado por três componentes: *(i)* um preço fixo P_i, *(ii)* um tempo de resposta T_i, e *(iii)* um conjunto de *templates* de consulta $\{M_{i1}, M_{i2}, \ldots, M_{iv}\}$ [Ortiz et al. 2015]. Os *templates* de consulta delimitam o conjunto de operações que o usuário pode executar em cada *tier*, enquanto o limiar T_i garante que toda consulta compatível com esses *templates* seja respondida no tempo estipulado. Uma propriedade importante do PSLA é que cada *tier*

corresponde a um ponto distinto em um espaço multidimensional, definido por capacidades de consulta, desempenho e custo. Dessa forma, o usuário seleciona exatamente um *tier* do conjunto e, a partir dessa escolha, passa a operar com o preço, o desempenho e o conjunto de consultas previamente estabelecidos. [Ortiz et al. 2015] geram todas as possíveis configurações de parâmetros, que são agrupadas em *clusters* por tipos de consultas. Após essa etapa, consultas *skyline* são executadas para definir os PSLA finais. É importante destacar que, como a *skyline* é aplicada, *tiers* potencialmente úteis podem ser descartados, uma vez que o operador *skyline* retornaapenas *um* resultado.

Consultas *k-Skyband*. A noção de *dominância* entre vetores multidimensionais é a base dos operadores de consulta de preferência em bancos de dados [Borzsony et al. 2001]. Considere um conjunto de objetos $\{p_1, p_2, \ldots, p_N\}$ definidos em um espaço d-dimensional. Diz-se que p_i *domina* p_j ($p_i \prec p_j$) se, e somente se, p_i não apresenta valor superior ao de p_j em nenhuma dimensão e é estritamente inferior em pelo menos uma [Papadias et al. 2005]. Por definição, consultas *k-Skyband* retornam o conjunto de pontos dominados por, no máximo, k outros pontos. Assim, para $k = 1$, o conjunto resultante corresponde à consulta *skyline*; para $k > 1$, camadas adicionais de dominância são incorporadas progressivamente, formando o *k-Skyband* [Ciaccia and Martinenghi 2020]. O resultado de uma consulta *k-Skyband* contém o conjunto de respostas de qualquer consulta *top-k* baseada em uma função de pontuação monotônica arbitrária [Siddique et al. 2018]. Isso implica que o *k-Skyband* pode ser utilizado como etapa de pré-processamento para consultas *skyline* e *top-k*, reduzindo o espaço de busca sem comprometer a qualidade do resultado final. De acordo com [Bedo et al. 2019, Ciaccia and Martinenghi 2020, Ciaccia and Martinenghi 2025], essa propriedade pode ser generalizada por meio do conceito de $\mathcal{F}$-dominância, no qual a relação de dominância é avaliada em relação a um conjunto arbitrário de funções monotônicas $\mathcal{F}$. No contexto desse estudo, consultas *k-Skyband*, originalmente propostas na literatura sobre bancos de dados como uma generalização da dominância de Pareto, mostram-se úteis para identificar configurações dos *workflows* de treinamento de modelos que são dominadas por até k outras configurações no espaço multicritério [Gao et al. 2014].

Trabalhos Relacionados. Alguns trabalhos na literatura têm como foco facilitar a definição de SLAs em ambientes de computação em nuvem com múltiplos objetivos. [Ortiz et al. 2015] foram pioneiros na proposição de SLAs personalizados, com o objetivo de simplificar a configuração e a precificação de sistemas de banco de dados em nuvem. O trabalho de [Ortiz et al. 2015] serve de base para o modelo proposto neste artigo, que estende esse conceito para *workflows* de treinamento de modelos de aprendizado de máquina, considerando um conjunto mais amplo de métricas de otimização e o uso de consultas *k-Skyband* na geração de *tiers*. Entretanto, outros trabalhos também investigam abordagens para a construção de SLAs mais acessíveis e flexíveis, propondo diferentes estratégias para apoiar usuários na definição de acordos de nível de serviço em cenários complexos. [Boukadi et al. 2016] propõem um método orientado a métricas de Qualidade de Serviço (QoS) para a definição automática de SLAs em ambientes de nuvem. A abordagem parte de uma especificação, fornecida pelo usuário, dos requisitos do *workflow* e de suas restrições de QoS, bem como de um conjunto de recursos disponíveis na nuvem. Com base nessas informações, o SLA é gerado automaticamente por meio de uma abordagem baseada em *Model-Driven Architecture* (MDA), garantindo a consistência e a correção das especificações. Apesar de eficiente, o método pressupõe que o usuário seja capaz de explicitar todas as restrições de QoS, o que pode representar um desafio em cenários reais.

[Venkataraman et al. 2016] propõem uma abordagem para criação de SLAs a partir da predição de desempenho em aplicações de análise de dados em larga escala, explorando a observação de que *jobs* apresentam características semelhantes. A partir dessa premissa, os autores constroem modelos de desempenho com base na execução de amostras reduzidas de dados, que são posteriormente utilizados para estimar o comportamento das aplicações em conjuntos de dados maiores e em diferentes configurações de *cluster*, bem como para definir os SLAs. Apesar de eficaz na predição de desempenho, o trabalho não aborda explicitamente como os SLAs são criados nem considera múltiplos critérios. [Gouryraj et al. 2021] propõem dois modelos de aprendizado de máquina para previsão de SLAs. O primeiro modelo utiliza técnicas de regressão para estimar o tempo necessário para a resolução de chamados (*Estimated Time to Resolve*), a partir do qual o valor do SLA é derivado. O segundo modelo adota uma abordagem de classificação, responsável por predizer diretamente o valor do SLA. Apesar de interessante, os SLAs gerados focam em somente um critério, o tempo de resposta de chamados, o que difere da abordagem por dominância abordada nesse estudo.

3. A Abordagem `PSLA4ML`

Esta seção apresenta os detalhes do `PSLA4ML`, incluindo a formalização do modelo e o processo de geração dos *tiers*.

3.1. Formalização do `PSLA4ML`

O conceito de PSLAs foi proposto no contexto de SGBDs em nuvem como um mecanismo para auxiliar os usuários a definir SLAs que garantam, simultaneamente, desempenho e custo para diferentes classes de consultas SQL. Na proposta original, as consultas submetidas são agrupadas em classes representadas por *templates*, e cada classe é associada a *tiers* que especificam limites de tempo de execução e respectivos custos financeiros. A ideia por trás dos PSLAs pode ser estendida a *workflows* de treinamento de modelos de ML, que apresentam características heterogêneas e demandas de recursos variáveis. Em geral, os *workflows* de treinamento diferem quanto ao tipo de modelo, aos valores dos hiperparâmetros, ao tamanho do conjunto de dados e ao número de iterações.

Neste estudo, estendemos o conceito de PSLA por meio de uma estratégia alternativa, baseada em consultas a bancos de dados de proveniência [Herschel et al. 2017], para caracterizar *workflows* reais de treinamento de modelos de ML. Esses bancos armazenam metadados detalhados de execuções anteriores desses *workflows*, incluindo informações sobre as configurações dos modelos, os conjuntos de dados utilizados, os recursos computacionais empregados e as métricas de desempenho obtidas. Esses registros fornecem uma base sólida para a análise e a modelagem do comportamento de *workflows*. Ressalta-se que a captura desses dados está fora do escopo deste artigo, sendo delegada a abordagens como o Cerebro [Nakandala et al. 2020] ou a DLProv [Pina et al. 2024, Pina et al. 2025].

Assim, propõe-se o `PSLA4ML` como uma adaptação do conceito de PSLA ao domínio de ML, em que a unidade básica de análise deixa de ser a consulta SQL e passa a ser o *workflow* de treinamento. Nesse contexto, introduzimos a noção de *training template* como uma abstração capaz de agrupar *workflows* com características estruturais semelhantes. Formalmente, um *training template* é definido como uma tupla $\mathcal{T} = \langle A, D, H \rangle$, em que: (i) A denota a arquitetura do modelo (*e.g.*, *Transformer*, CNN, *Gradient Boosted Trees*); (ii) D representa o conjunto de dados de entrada, caracterizado por seu identificador, dimensionalidade e tamanho; e (iii) H é o espaço de hiperparâmetros *fixos* do *workflow*, *i.e.*, aqueles que definem a

topologia do modelo e não variam entre execuções do mesmo *template* (*e.g.*, tipo de função de *loss*, número de camadas, tipo de *tokenizer*). Dois *workflows* W_1 e W_2 são considerados instâncias do mesmo *training template* $\mathcal{T}$ se e somente se compartilham os mesmos A, D e H. Os hiperparâmetros de otimização, *e.g.*, *learning rate*, *batch size*, otimizador e *dropout*, são tratados como variáveis livres dentro de um *template*, constituindo o espaço de busca explorado. Essa separação entre atributos estruturais fixos e hiperparâmetros variáveis permite que o PSLA4ML recupere *traces* historicamente relevantes e produza *tiers* comparáveis entre si. No experimento conduzido neste artigo, por exemplo, um *template* corresponde a $\mathcal{T} = \langle$BERT-PLI, COLLIE, dropout = 0,1$\rangle$. A partir desses *templates*, torna-se possível consultar bancos de dados de proveniência para recuperar execuções passadas de treinamentos semelhantes, permitindo explorar configurações e analisar seu comportamento histórico.

Cada execução armazenada no banco de proveniência, *i.e.*, cada *trace*, reúne informações sobre a configuração do treinamento, os recursos computacionais utilizados e as métricas observadas durante sua execução. Entre essas informações, destacam-se as características do modelo e do conjunto de dados, os recursos alocados (*e.g.*, CPU, GPU e memória), os hiperparâmetros empregados, bem como as métricas de tempo de execução, custo financeiro, consumo energético e emissão de carbono (denominadas por t, c_f, c_e, c_c respectivamente). Em conjunto, esses *traces* formam um espaço multidimensional de alternativas/*rankings* que expressa diferentes compromissos entre desempenho, custo e sustentabilidade. Para identificar configurações promissoras para a definição de *tiers*, utilizamos consultas baseadas no operador *k-Skyband*, que permitem selecionar execuções representativas mesmo em cenários com múltiplos critérios de interesse.

Conforme apresentado na definição da Seção 2, o operador *k-Skyband* é uma generalização do operador *Skyline* e permite identificar pontos dominados por, no máximo, k outros pontos no espaço de atributos considerado. Intuitivamente, isso permite selecionar configurações que representam bons compromissos entre múltiplos objetivos, evitando soluções sub-ótimas. Considerando um conjunto de execuções registradas $P = \{p_i, \ldots, p_n\}$, definimos o conjunto *k-Skyband* como $Skyband_k(P) = \{p \in P; |\{q \in P : q \prec p\}| \leq k\}$, onde $q \prec p$ indica que q domina p segundo os critérios de otimização considerados (*e.g.*, tempo de execução, emissão de carbono, *etc*). Os pontos do conjunto *k-Skyband* representam configurações que oferecem bons compromissos entre os objetivos. A partir desses pontos, é possível derivar automaticamente diferentes *tiers*. Cada ponto selecionado pode ser interpretado como um candidato a *tier*, caracterizado por uma configuração específica de recursos e por limites de desempenho esperados. Assim, o PSLA4ML pode ser definido como um conjunto de níveis de serviço derivados do conjunto *k-Skyband*, $PSLA = \{R_1, R_2, \ldots, R_k\}$, em que cada nível R_i corresponde a uma configuração não dominada ou k-não dominada identificada a partir da análise do banco de dados de *traces* de proveniência.

A criação dos *tiers* com o operador *k-Skyband* apresenta três principais vantagens. Primeiro, permite que os *tiers* reflitam o comportamento observado em *workflows* reais, o que torna as estimativas de desempenho mais fidedignas. Em segundo lugar, o uso de consultas *k-Skyband* permite capturar diferentes compromissos entre os objetivos, oferecendo múltiplos níveis de serviço adaptados aos requisitos dos *stakeholders*. Terceiro, as consultas *k-Skyband* têm custo polinomial conhecido e podem ser implementadas de forma distribuída, beneficiando-se do ambiente de nuvem. Dessa forma, a combinação de dados de proveniência e consultas *k-Skyband* fornece um mecanismo para derivar *tiers* em plataformas de treino de modelos de ML, eliminando a geração artificial de *workloads* e explorando o histórico.

3.2. Processo de Geração de *Tiers*

A Figura 1 apresenta uma visão geral do processo proposto para a geração de *tiers* a partir de dados de proveniência. O processo é composto por cinco etapas, sendo as duas primeiras opcionais: *(i) Grid Search*, *(ii)* execução dos *workflows* de treinamento, *(iii)* extração dos *traces*, *(iv)* cálculo de métricas derivadas e *(v)* identificação dos *tiers*. A etapa de *Grid Search* é executada apenas quando o banco de dados de proveniência não está disponível para consulta. Nesse cenário, são geradas as possíveis combinações de hiperparâmetros, de dados de entrada e de ambientes computacionais a serem exploradas na construção dos *tiers*. Ressalta-se que, dependendo dos valores atribuídos a cada hiperparâmetro, o número de combinações pode aumentar significativamente. Atualmente, essa etapa é implementada por meio de um *script* em Python que utiliza *ProcessPoolExecutor*, em conjunto com o PyTorch, para gerenciar e gerar as combinações em paralelo.

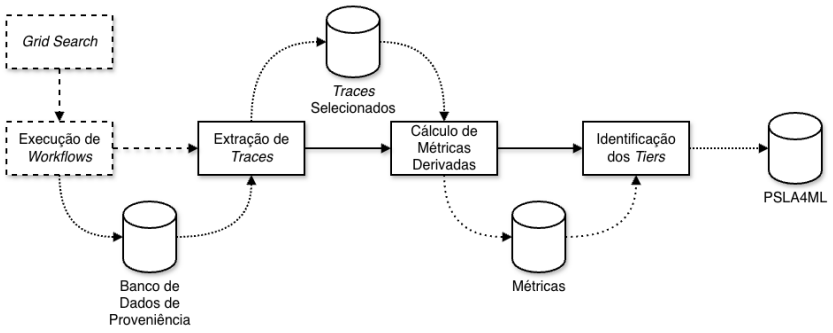

Figura 1. Visão geral do processo de geração dos *tiers* do PSLA4ML.

Vale ressaltar que a necessidade do *Grid Search* no presente artigo decorre de uma condição de *cold start*, *i.e.*, na ausência de um banco de proveniência pré-existente para o *training template* em questão, é preciso popular esse banco com um conjunto mínimo de *traces* antes que consultas k-Skyband possam ser executadas. Esse comportamento é análogo ao problema de *cold start* em sistemas de recomendação [Peoples et al. 2022], em que a ausência de histórico impede a personalização imediata. Uma vez que o banco de proveniência contenha *traces* suficientes para o template, provenientes de execuções anteriores monitoradas por ferramentas como o Cerebro [Nakandala et al. 2020] ou a DLProv [Pina et al. 2024], a etapa de *Grid Search* não é executada e o processo opera apenas sobre dados históricos reais.

A segunda etapa corresponde à execução dos *workflows* e é responsável por processar todas as combinações de *workflows* de treinamento geradas na etapa de *Grid Search*. Essa etapa é implementada por meio de um *script* em Python que orquestra a execução dos *workflows* em dois modos de operação, selecionáveis por meio de argumentos na linha de comando. O primeiro modo corresponde à execução isolada de experimentos (*-mode single*), enquanto o segundo refere-se à execução em grade (*-mode grid*). No modo *single*, o *script* invoca diretamente a função *execute_experiment()* para executar especificamente um *workflow* ou para validar pontualmente uma combinação de configurações. Já no modo *grid*, a orquestração segue as definições estabelecidas na etapa de *Grid Search*. É importante ressaltar que assumimos que toda execução do *workflow* é monitorada por meio de uma abordagem de captura de dados e que os metadados da execução são salvos no banco de dados de proveniência.

A etapa de extração dos *traces* consiste na seleção dos *traces* de proveniência armazenados no banco de dados que são relevantes para a definição dos *tiers*. Essa seleção é importante, pois a base pode conter *traces* muito antigos, associados a execuções que já não

refletem o contexto de uso atual. Além disso, determinados *traces* podem estar relacionados a experimentos que não são mais executados pelo usuário ou a infraestruturas computacionais que já não estão disponíveis (*e.g.*, um *cluster* que era usado no passado, mas não está mais disponível). Dessa forma, a filtragem adequada desses dados contribui para garantir a consistência e a relevância das informações utilizadas na construção dos *tiers*. Ao final, cada *trace* extraído possui as seguintes informações: *(i) Workflow*: UUID, *dataset* de entrada, dimensionalidade e tamanho do *dataset*, status e *timestamps* de início e fim, *(ii)* Ambiente de Execução: tipo e nome do dispositivo, precisão numérica, *(iii)* Hiperparâmetros: otimizador, *learning rate*, *batch size* e número de épocas, e *(iv)* Métricas: tempo de execução (s), custo (USD), RAM média e pico (MB), F1-*score* e acurácia.

Entretanto, algumas métricas não precisam ser obtidas diretamente do banco de dados de proveniência, pois não são computadas automaticamente. Essas métricas são derivadas a partir de dados já disponíveis, incluindo: *(i)* consumo de energia (kWh) e *(ii)* emissões de CO_2 (kg). Para o cálculo do consumo energético e das emissões de CO_2, utilizamos a biblioteca CodeCarbon [Luccioni et al. 2023], que fornece estimativas detalhadas por execução de *workflow* com base em informações como tempo de execução, tipo de dispositivo e outros. Essas métricas são utilizadas de forma complementar a outras já consideradas, como tempo de execução, custo financeiro, acurácia e F1-*score*, na geração dos *tiers*. Todas essas métricas são adicionadas aos *traces* extraídos anteriormente.

Por fim, com os *traces* completos à disposição, a consulta *k-Skyband* é realizada. Para cada *trace* retornado pela consulta *k-Skyband*, realiza-se a discretização das métricas contínuas, de modo que os *tiers* sejam associados a intervalos, e não a valores exatos, o que se mostra mais realista do ponto de vista da previsão de recursos. Ressalta-se, ainda, que, para a definição dos *tiers*, o usuário pode selecionar quais métricas serão consideradas. Por exemplo, se o objetivo for definir SLAs para que *workflows* executem de forma mais eficiente, as métricas de desempenho devem ser priorizadas. Por outro lado, se o foco estiver na qualidade dos resultados, independentemente de desempenho, métricas como o F1-*score* e a acurácia podem ser adotadas. O Algoritmo 1 sintetiza a geração dos *tiers* de acordo com a abordagem proposta. O código-fonte para a geração do `PSLA4ML` pode ser obtido no repositório: `https://github.com/gassantos/gridsearch-skyband`.

Algoritmo 1 Geração de *Tiers* a partir de dados de proveniência usando *k-Skyband*

Entrada: Banco de dados de proveniência P, parâmetro k
Saída: Conjunto de *tiers* T
1: Extrair registros de execução (*traces*) do banco de dados de proveniência
2: Extrair métricas de desempenho (t, c_f) para cada tupla
3: Calcular métricas de desempenho derivadas (c_c, c_e) para cada tupla
4: $S \leftarrow Skyband_k(P)$
5: **para** cada configuração p em S **faça**
6: Criar *tier* R
7: Adicionar R ao conjunto T
8: **fim para**
9: Discretizar métricas contínuas em T
10: **retorne** T

4. Avaliação Experimental

Esta seção tem como objetivo avaliar a viabilidade da abordagem proposta em um cenário de treinamento de modelos em condições reais.

Configuração do Experimento. Para analisar a viabilidade da geração dos *tiers* no `PSLA4ML`, escolhemos um *workflow* baseado no BERT (*Bidirectional Encoder Representations from Transformers*) adaptado para a tarefa de *Paraphrase Language Inference* (PLI) [Devlin et al. 2019]. Nesse contexto, o modelo é utilizado para avaliar o grau de equivalência semântica entre pares de textos. O BERT é um modelo de linguagem baseado na arquitetura de *Transformers*, capaz de analisar o contexto das palavras de forma bidirecional, *i.e.*, considerando simultaneamente as palavras à esquerda e à direita de cada termo em uma sentença. Para este experimento, utilizamos textos do conjunto de dados COLLIE [Yao et al. 2023], que disponibiliza dados sintéticos compostos por casos jurídicos. Como não havia, inicialmente, um banco de dados de proveniência contendo *traces* de execuções anteriores de *workflows* de treinamento, realizou-se um *Grid Search* com um conjunto reduzido de combinações de hiperparâmetros. As variações exploradas incluem dois valores de taxa de aprendizado (*learning rate*) $\{1\times10^{-5}, 2\times10^{-5}\}$, dois tamanhos de *batch* $\{8, 16\}$, três otimizadores distintos {*Adam*, *AdamW*, *BertAdam*}, e um valor fixo de *dropout* $\{0.1\}$. Todas as execuções do *workflow* foram realizadas em paralelo, com, no máximo, dois *workers*. Esse conjunto de variações permite explorar, de forma controlada, o espaço de busca de hiperparâmetros, possibilitando a análise do impacto dessas variáveis sobre o desempenho, o custo e o consumo de recursos durante a execução do *workflow*.

Configuração do Ambiente. Os experimentos foram conduzidos em três configurações de ambiente computacional na nuvem do Google, abrangendo execuções em CPU, GPU e TPU, com o objetivo de avaliar o comportamento dos *workflows* sob diferentes perfis de *hardware*. Em todos os cenários, foi utilizada a mesma pilha de *software*, com Python 3.12.12 e SO Linux (kernel 6.6.113+, arquitetura x86_64, glibc 2.35). No caso da GPU, utilizou-se uma NVIDIA RTX PRO 6000 com suporte a CUDA 12.8 e 95 GB de memória de vídeo, em conjunto com um processador AMD EPYC 9B45 (24 núcleos) e 176 GB de RAM. O ambiente baseado em TPU utilizou um acelerador TPU v6e (v6e-1) acoplado a um processador AMD EPYC 9B14 (22 núcleos) e 172 GB de RAM. Por fim, o ambiente de CPU contou com um processador AMD EPYC 7B12 (4 núcleos) e 50 GB de RAM. Considerando o ambiente de execução no Google, execuções em CPU apresentam custo por hora reduzido de US$ 0,10. Por outro lado, execuções em GPU apresentam custos na faixa de aproximadamente US$1,20/h para configurações de alto desempenho similares à utilizada neste artigo. Já o uso de TPU (v6e) apresenta custo de US$1,50/h. A escolha desses três ambientes foi motivada pela necessidade de capturar diferentes *trade-offs* entre desempenho, custo e eficiência energética. Enquanto GPUs são amplamente utilizadas no treinamento de modelos como o BERT, TPUs oferecem otimizações específicas para operações de aprendizado profundo, o que pode proporcionar ganhos de eficiência energética. Por outro lado, a execução em CPU representa um cenário mais acessível e disponível a mais pesquisadores, embora com desempenho relativo menor. Dessa forma, a análise comparativa entre esses ambientes contribui para uma compreensão mais abrangente do impacto da infraestrutura computacional na geração dos *tiers*.

Análise Preliminar. Antes da geração dos *tiers*, realizou-se uma análise preliminar dos *traces*, com o objetivo de validar os *tiers* obtidos. A Tabela 1 apresenta a média dos tempos de execução do *workflow* de treinamento em cada ambiente computacional, para cada otimizador. Como esperado, a GPU apresentou o melhor desempenho, com tempos variando entre 30,29s e 36,41s (média de 32,93s em todos os cenários). Por outro lado, as execuções em CPU demandaram entre 5.410,80s e 6.793,03s (média de 6.336,39s), enquanto as em TPU demandaram entre 3.634,15s e 3.701,62s (média de 3.666,25s).

O *speedup* da GPU em relação à CPU variou entre 148,3× (com o otimizador *AdamW*) e 182,3× (com o *BertAdam*). Em comparação com a TPU, a GPU foi de 100,5× a 123,2× mais rápida. Por sua vez, a TPU apresentou ganho modesto, entre 1,46× e 1,48×, em relação à CPU. Esse resultado merece atenção: a análise das emissões de carbono por meio da biblioteca CodeCarbon revelou que, em alguns experimentos com TPU, o campo `gpu_power` foi registrado como nulo (0,0 W), enquanto o campo `device_type` foi identificado como `CPU`, com arquitetura `x86_64`. Isso indica que o *workflow* foi executado integralmente na CPU hospedeira (*host CPU*) do ambiente de TPU, sem delegação efetiva ao acelerador. No que tange à escolha do otimizador, o *BertAdam* foi o mais eficiente em termos de tempo, sendo mais rápido em dois dos três ambientes, atingindo os tempos de treinamento mais baixos de 3.634,15s (TPU) e 30,29s (GPU).

Tabela 1. Tempo total de treinamento (s) por ambiente e otimizador.

Otimizador	CPU (s)	TPU (s)	GPU (s)
Adam	5.410,80	3.701,62	36,41
AdamW	6.805,34	3.662,99	32,10
BertAdam	6.793,03	3.634,15	30,29
Média	**6.336,39**	**3.666,25**	**32,93**

As métricas de consumo energético e de emissões de carbono foram calculadas por meio da ferramenta CodeCarbon, e suas médias são apresentadas na Tabela 2. Observa-se que a execução em GPU apresentou consumo médio de $1{,}69 \times 10^{-3}$ kW por experimento, o que representa uma redução de 99,2% em relação à CPU (média de 0,2463 kW) e de 98,1% em relação à TPU (média de 0,0941 kW), mesmo considerando o maior custo energético por hora associado ao uso de GPUs (a CPU consome 0,1 kW/h, a GPU 0,45 kW/h e a TPU 0,35 kW/h). Esse resultado está diretamente relacionado à diferença significativa no tempo de execução entre os ambientes, uma vez que as execuções em CPU e TPU apresentaram tempos de ordem de grandeza superiores. Assim, apesar do maior consumo instantâneo de energia, a GPU se mostra mais eficiente em termos de energia total consumida, devido à redução no tempo necessário para a conclusão dos experimentos.

Tabela 2. Consumo energético e emissões de CO_2 por otimizador e ambiente

Ambiente	Otimizador	Energia (kWh)	CO_2 (kg)
CPU	Adam	0,210364	0,056298
CPU	AdamW	0,264602	0,070813
CPU	BertAdam	0,264134	0,070688
TPU	Adam	0,095077	0,025445
TPU	AdamW	0,094093	0,025181
TPU	BertAdam	0,093357	0,024984
GPU	Adam	0,001847	0,000836
GPU	AdamW	0,001652	0,000748
GPU	BertAdam	0,001577	0,000714

Os experimentos, independentemente dos otimizadores adotados, apresentaram acurácia máxima de 0,5 em todas as configurações avaliadas. Esse resultado indica que o conjunto de hiperparâmetros considerado nesta varredura inicial é insuficiente para promover uma generalização adequada. Ressalta-se, contudo, que tal comportamento não é inesperado, uma vez que os experimentos envolveram um espaço reduzido de variações de hiperparâmetros, selecionados com o objetivo principal de avaliar a viabilidade e a construção do `PSLA4ML`, e não de otimizar o desempenho preditivo do modelo.

Análise dos PSLA4ML Gerados. Após a análise preliminar, foram definidos os *tiers* associados aos PSLA4ML. Nessa etapa, optou-se por não considerar as métricas de F1-*score* e acurácia, uma vez que a varredura de hiperparâmetros não teve como objetivo a obtenção de modelos com desempenho preditivo ótimo. Em vez disso, a análise concentrou-se em métricas relacionadas à eficiência e ao custo do *workflow*, a saber: *(i)* tempo de execução do *workflow* (*train_time_sec*), *(ii)* consumo de energia (*energy_kwh*), *(iii)* emissões de CO_2 (*emissions_kg_co2*) e (iv) custo em dólares (*cost_usd*). Adicionalmente, o parâmetro k da consulta k-*Skyband* foi ajustado para $k = 2$ e $k = 3$. Do ponto de vista das consultas k-*Skyband*, o conjunto de experimentos realizado define um espaço de pontos no hiperplano $\langle$*train_time_sec*, *energy_kwh*, *emissions_kg_co2*, *cost_usd*$\rangle$, no qual cada configuração (*hardware* $\times$ hiperparâmetros) é tratada como um ponto candidato. A Tabela 3 apresenta os *tiers* identificados a partir da execução da consulta k-*Skyband* para diferentes configurações do parâmetro k, especificamente $k = 1$ (equivalente à consulta *Skyline*), $k = 2$ e $k = 3$. Cada configuração resulta em um conjunto distinto de soluções não dominadas ou parcialmente dominadas, refletindo diferentes níveis de relaxamento no critério de dominância. Dessa forma, enquanto $k = 1$ retorna apenas as soluções estritamente não dominadas, valores maiores de k permitem a inclusão de configurações que, embora dominadas por um número limitado de alternativas, ainda representam compromissos relevantes entre as métricas consideradas.

Observa-se um padrão consistente de *trade-off* entre desempenho computacional e custo financeiro. Para $k = 1$, as configurações utilizam hiperparâmetros semelhantes, diferindo apenas no *hardware*. A execução em GPU apresenta menor tempo (< 5000), menor consumo energético (< 0.2) e menores emissões de CO_2 (< 0.05), porém com maior custo (≥ 1.2). Em contrapartida, a execução em CPU implica maior tempo (≥ 5000) e maior impacto ambiental, mas menor custo (< 1.2), evidenciando a maior eficiência das GPUs em cenários intensivos. Para $k = 2$, observa-se variação nas configurações, especialmente no otimizador e no *batch size*. A configuração com GPU (*batch*=16 e *BertAdam*) mantém melhor desempenho em tempo e energia, enquanto a configuração em CPU (com *AdamW*) apresenta maior custo computacional, porém menor custo financeiro. Isso indica que o aumento do *batch size*, aliado ao uso de GPU, favorece a eficiência. Para $k = 3$, o mesmo padrão se mantém. A configuração com GPU (*batch*=16 e *AdamW*) apresenta melhores resultados de tempo, energia e emissões, enquanto a configuração com CPU (*batch*=8 e *Adam*) reduz custo financeiro às custas de pior desempenho e maior impacto ambiental.

Tabela 3. PSLA4ML gerados para diferentes valores de k.

k	Mod.	Dataset	LR	Batch	Opt.	Drop	HW	*Tempo*	*Energia*	CO_2	*Custo*
$k = 1$	BERT-PLI	COLLIE	1×10^{-5}	8	BertAdam	0.1	GPU	< 5000	< 0.2	< 0.05	≥ 1.2
	BERT-PLI	COLLIE	1×10^{-5}	8	BertAdam	0.1	CPU	≥ 5000	≥ 0.2	≥ 0.05	< 1.2
$k = 2$	BERT-PLI	COLLIE	1×10^{-5}	8	AdamW	0.1	CPU	≥ 5000	≥ 0.2	≥ 0.05	< 1.2
	BERT-PLI	COLLIE	1×10^{-5}	16	BertAdam	0.1	GPU	< 5000	< 0.2	< 0.05	≥ 1.2
$k = 3$	BERT-PLI	COLLIE	1×10^{-5}	16	AdamW	0.1	GPU	< 5000	< 0.2	< 0.05	≥ 1.2
	BERT-PLI	COLLIE	1×10^{-5}	8	Adam	0.1	CPU	≥ 5000	≥ 0.2	≥ 0.05	< 1.2

A Figura 2 ilustra a distribuição dos *traces* obtidos ao longo dos experimentos, bem como a localização dos pontos selecionados para a definição dos *tiers* dos PSLA4ML. Observa-se que os *tiers* são compostos por subconjuntos representativos desses *traces*, escolhidos com base em critérios de eficiência e custo, conforme modelados pelas métricas analisadas. Cabe destacar que o objetivo dos *tiers* não é abranger exaustivamente todas as possíveis configura-

ções para usuários interessados em treinar novos modelos. Em vez disso, a proposta consiste em oferecer um conjunto reduzido e qualificado de alternativas, selecionadas conforme o critério de dominância estabelecido pelas consultas *k-Skyband*. Essa abordagem permite apresentar opções que representam diferentes compromissos (*trade-offs*) entre as métricas consideradas, facilitando a tomada de decisão por parte do usuário sem sobrecarregá-lo com um espaço excessivamente amplo de possibilidades.

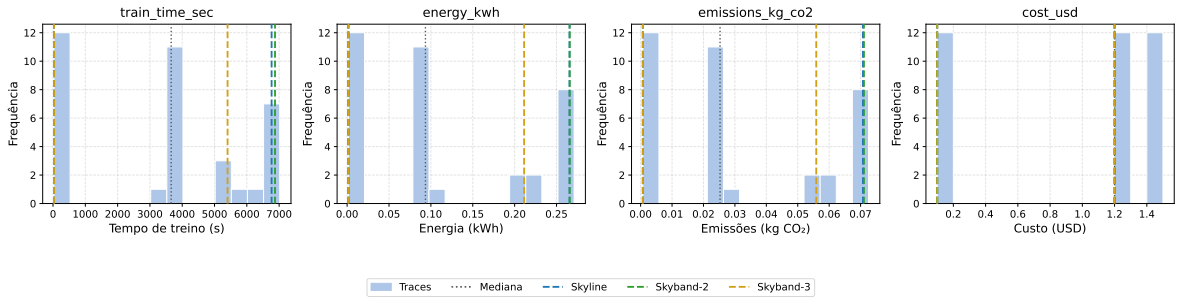

Figura 2. Distribuição dos *traces* em função dos valores das métricas.

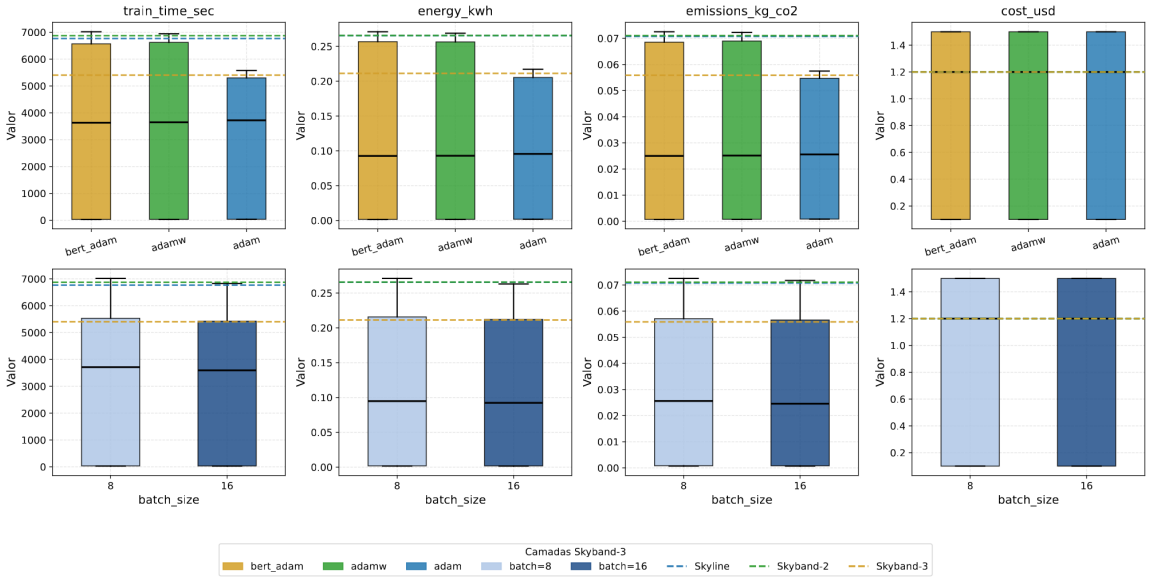

Figura 3. Distribuição das métricas por otimizador e por tamanho de *batch*.

A Figura 3 estratifica os atributos utilizados na consulta *k-Skyband* em função dos hiperparâmetros, permitindo identificar os fatores que mais influenciam a posição das configurações no espaço de dominância. A análise das métricas agregadas por otimizador indica diferenças relativamente sutis entre as configurações. De modo geral, *bert_adam* apresenta as menores medianas para *train_time_sec*, *energy_kwh* e *emissions_kg_co2*, seguido de perto por *adamw*, enquanto *adam* tende a apresentar valores ligeiramente superiores. Em termos de dispersão, observa-se maior variabilidade para *bert_adam* e *adamw*, enquanto *adam* apresenta comportamento mais estável (menor IQR). Para *cost_usd*, as diferenças são pouco expressivas entre os otimizadores, indicando baixa sensibilidade dessa métrica à escolha do método de otimização. Esses resultados sugerem que, embora existam diferenças, o impacto do otimizador sobre as métricas analisadas é moderado. No caso do *batch size*, as diferenças são ainda mais discretas. As medianas de *train_time_sec*, *energy_kwh* e *emissions_kg_co2*

são muito próximas entre os valores analisados, com leve vantagem para configurações com *batch* menor em tempo de execução. A dispersão também se mantém semelhante entre os grupos, indicando comportamento consistente independentemente do tamanho do *batch*. Para *cost_usd*, novamente não há distinção relevante. Em conjunto, esses resultados indicam que o *batch size* exerce influência limitada nas métricas consideradas, sendo menos determinante que o otimizador na diferenciação das configurações no espaço de dominância.

É importante ressaltar que as métricas *train_time_sec*, *energy_kwh*, *emissions_kg_co2*, *cost_usd* formam um conjunto de alta colinearidade, com coeficientes $r > 0{,}99$ em todos os pares de métricas. Esse comportamento decorre do fato de que, em ambiente de *hardware* fixo, o consumo energético é proporcional ao tempo de execução, e as estimativas de emissões e custo são derivadas diretamente do tempo e do consumo energético. Do ponto de vista das consultas *k-Skyband*, essa multicolinearidade tem implicações diretas na estrutura da fronteira, *i.e.*, em espaços com atributos altamente correlacionados, o conjunto *Skyline* tende a colapsar em poucos pontos extremos, pois minimizar qualquer atributo do grupo implica, quase igualmente, minimizar os demais. Dois fatores limitam a generalização dos resultados aqui apresentados: (i) O ambiente TPU não utilizou efetivamente o acelerador, *i.e.*, os *logs* do CodeCarbon indicam *gpu_power* = 0,0 W e *device_type* = *CPU*, sugerindo ausência de execução na MXU. (o ganho de $\approx 1{,}47\times$ em relação à CPU reflete diferenças entre *hosts*, e não o potencial da TPU), e (ii) O espaço de hiperparâmetros foi intencionalmente reduzido (poucas configurações por ambiente, com *learning rate* e *batch size* fixos), priorizando a análise do `PSLA4ML`. Para conclusões sobre qualidade preditiva, é necessária uma exploração mais ampla, incluindo variações de *max_seq_length*, taxa de aprendizado e número de épocas.

5. Conclusões e Trabalhos Futuros

Este estudo apresentou o `PSLA4ML`, uma extensão do conceito de PSLA para o contexto de treino de modelos de ML, utilizando dados de proveniência e consultas *k-Skyband* para derivar *tiers* de serviço. A abordagem proposta permite capturar, de forma integrada, diferentes métricas relevantes, *e.g.*, tempo de execução, custo financeiro, consumo energético e emissões de carbono, oferecendo suporte à definição de SLAs em ambientes de nuvem. Os resultados indicaram a viabilidade da proposta e evidenciaram padrões consistentes de *trade-off* entre as métricas analisadas. Em particular, observou-se que o tipo de *hardware* é o principal fator, com execuções em GPU apresentando maior eficiência computacional e energética, enquanto as em CPU se destacam pelo menor custo financeiro. As consultas *k-Skyband* mostraram-se eficazes na identificação de configurações, reduzindo o espaço de busca a um conjunto enxuto de alternativas relevantes. Além disso, verificou-se que o impacto de hiperparâmetros como otimizador e *batch size* é secundário em comparação ao ambiente computacional, e que as métricas analisadas apresentam alta correlação, o que influencia a estrutura das fronteiras de dominância. Como trabalhos futuros, destacam-se a ampliação do espaço de busca de hiperparâmetros (tanto do modelo de ML quanto do parâmetro de tolerância da consulta *k*-Skyband), a inclusão de métricas de qualidade do modelo nos critérios de análise e a aplicação do `PSLA4ML` a outros *workflows* e a cenários reais de *ML as a Service*.

Agradecimentos

Os autores agradecem à CAPES, ao CNPq e à Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro (E-26/204.238/2024 e E-26/204.544/2024) pelo apoio e financiamento parcial a essa pesquisa. Os autores utilizaram LLMs para revisão gramatical do texto do artigo.

Referências

Amorim, A., Moraes, J. V., Pina, D., Paes, A., and de Oliveira, D. (2026). On provenance in model fusion. In *SBBD 2026 - Short Papers*.

Bedo, M., Ciaccia, P., Martinenghi, D., and De Oliveira, D. (2019). A k–skyband approach for feature selection. In *SISAP'19*, pages 160–168. Springer.

Borzsony, S., Kossmann, D., and Stocker, K. (2001). The skyline operator. In *Proceedings 17th international conference on data engineering*, pages 421–430. IEEE.

Boukadi, K., Grati, R., and Ben-Abdallah, H. (2016). Toward the automation of a qos-driven sla establishment in the cloud. *Service Oriented Computing and Applications*, 10(3):279–302.

Ciaccia, P. and Martinenghi, D. (2020). Flexible skylines: Dominance for arbitrary sets of monotone functions. *ACM Transactions on Database Systems (TODS)*, 45(4):1–45.

Ciaccia, P. and Martinenghi, D. (2025). Optimization strategies for parallel computation of skylines. *Distributed and Parallel Databases*, 43(1):10.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of the 2019 NAACL-HLT*, pages 4171–4186.

Gao, Y., Miao, X., Cui, H., Chen, G., and Li, Q. (2014). Processing k-skyband, constrained skyline, and group-by skyline queries on incomplete data. *Expert Systems with Applications*, 41(10):4959–4974.

Gouryraj, S., Kataria, S., and Swvigaradoss, J. (2021). Service level agreement breach prediction in servicenow. In *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 689–698.

Halperin, D., de Almeida, V. T., Choo, L. L., et al. (2014). Demonstration of the myria big data management service. In Dyreson, C. E., Li, F., and Özsu, M. T., editors, *SIGMOD'14*, pages 881–884. ACM.

Herschel, M., Diestelkämper, R., and Ben Lahmar, H. (2017). A survey on provenance: What for? what form? what from? *VLDB J.*, 26(6):881–906.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90.

Lu, J., Gong, P., Ye, J., Zhang, J., and Zhang, C. (2023). A survey on machine learning from few samples. *Pattern Recognition*, 139:109480.

Luccioni, A. S., Viguier, S., and Ligozat, A.-L. (2023). Estimating the carbon footprint of bloom, a 176b parameter language model. *J. Mach. Learn. Res.*, 24(1).

Nakandala, S., Zhang, Y., and Kumar, A. (2020). Cerebro: A data system for optimized deep learning model selection. *Proceedings of the VLDB Endowment*, 13(12).

Ortiz, J., de Almeida, V. T., and Balazinska, M. (2015). Changing the face of database cloud services with personalized service level agreements. In *CIDR'15*. www.cidrdb.org.

Papadias, D., Tao, Y., Fu, G., and Seeger, B. (2005). Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82.

Peoples, C., Moore, A., and Georgalas, N. (2022). Customer classification recommender to support personalised service level agreements across the internet of things. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE.

Pina, D., Chapman, A., Kunstmann, L., de Oliveira, D., and Mattoso, M. (2024). Dlprov: A data-centric support for deep learning workflow analyses. In *Proc. of the DEEM@SIGMOD 2024*, page 77–85, New York, NY, USA. ACM.

Pina, D. B., Kunstmann, L. N. O., Chapman, A., de Oliveira, D., and Mattoso, M. (2025). Dlprov: a suite of provenance services for deep learning workflow analyses. *PeerJ Comput. Sci.*, 11:e2985.

Siddique, M. A., Zaman, A., and Morimoto, Y. (2018). Selection of important sets by using k-skyband query for sets. *International Journal of Advanced Computer Science and Applications*, 9(4).

Talaei Khoei, T., Ould Slimane, H., and Kaabouch, N. (2023). Deep learning: systematic review, models, challenges, and research directions. *Neural Computing and Applications*, 35(31):23103–23124.

Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. (2016). Ernest: efficient performance prediction for large-scale advanced analytics. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, page 363–378, USA. USENIX Association.

Yao, S., Chen, H., Hanjie, A. W., Yang, R., and Narasimhan, K. (2023). Collie: Systematic construction of constrained text generation tasks.

Zhao, W. X. et al. (2023). A survey of large language models. *arXiv:2303.18223*, 1(2).