Desafio técnico - Tech for Humans

Dossiê de pesquisa

Gabriel Del Monte Schiavi Noda

---

1 Introdução e contextualização

1.1 Descrição da *API* atual

A arquitetura de produto atual para a extração de dados de documentos está baseada em uma *API* que emprega um processo em duas etapas, utilizando um *Large Language Model* (*LLM*). Como primeira etapa, o modelo realiza a interpretação e extração das informações do documento a ser analizado. Após isso, uma segunda chamada é feita para que os dados extraídos pela primeira inferência sejam formatados em um arquivo *JSON*.

A atual arquitetura introduz falhas que podem comprometer a viabilidade operacional e a experiência de usuário, podendo ser resumidas em três itens:

1. Latência: a necessidade de duas chamadas consecutivas duplica o tempo total de inferência da aplicação. Isso impacta diretamente a latência percebida pelo o usuário final, sendo um fator limitante para aplicações que exigem respostas em tempo mínimo. Além disso, a latência da aplicação pode variar significativamente em *APIs* alocadas em nuvem devido à carga e à alocação de recursos disponíveis, inviabilizando um tempo mínimo de processamento garantido.

2. Custo: o uso de modelos considerados de ponta para cada uma das chamadas resulta em um grande custo operacional, considerado desnecessário para a segunda tarefa, onde é feita apenas uma formatação com os dados extraídos pela primeira inferência.

3. Complexidade e limitações: em geral, *LLMs* puramente textuais perdem a informação espacial e relacional referente à estrutura do documento. Isto se torna um grande problema para lidar com documentos extensos e com layouts complexos, como é o caso de tabelas, formulários e múltiplos blocos de texto, impedindo a compreensão completa de documentos técnicos.

Deste modo, a ineficiência da arquitetura atual não é apenas um problema de escolha de modelo mas também um problema arquitetural causado pela separação entre a extração e a formatação da saída para o usuário, podendo ocasionar em falhas e perdas de informação visual do documento a ser analizado.

1.2 Objetivo da pesquisa e tese de solução

O principal objetivo desta pesquisa é projetar e validar uma solução alternativa de código aberto, *open-source*, e auto-hospedada, *self-hosted*, que busque minizar os problemas de latência e custo, ao mesmo tempo que, ao menos, mantenha os níveis de acurácia na extração de dados estruturados e complexos.

A teste central a ser apresentada é que a solução mais viável e de maior impacto envolve a transição para um Vision-Language Model (*VLM*) *end-to-end* de pequeno porte. Este modelo deve ser otimizado para a inferência de alta velocidade e especializado por meio de *fine-tuning* para garantir a compreensão de *layouts* complexos solicitados e para a geração do arquivo *JSON* estruturado em um único passo. Esta abordagem busca resolver, parcialmente ou completamente, os pilares de falhas da arquitetura atual.

2 Metodologia

2.1 Fontes de dados

A metodologia desta pesquisa buscou um rigor científico baseado em fontes técnicas e acadêmicas, especialmente em *sites* conhecidos no meio de inteligência artificial e *machine learning*, como por exemplo, *Hugging Face* e *Kaggle*.

As fontes principais da pesquisa incluíram publicações em repositórios como *arXiv* e *benchmarks* publicados em artigos no site *Hugging Face*. Vale ressaltar que o uso de ferramentas de IA, especialmente o Google Gemini, foram empregadas como apoio neste momento, facilitando a sumarização de artigos técnicos e a prototipagem de códigos Python para os experimentos realizados.

2.2 Estratégia de avaliação e principais modelos

A avaliação das soluções candidatas baseou-se em um conjunto de métricas empíricas e holísticas focadas na viabilidade de produção e no alinhamento com os objetivos do desafio.

Inicialmente, foram buscados modelos especializados na técnica de *Optical Character Recognition*, *OCR*, que não excedessem um total de 8 bilhões de parâmetros, afinal, os ambientes de testes considerados para este desafio possuíam limitações de *hardware* bem definidas, onde o ambiente *Google Colab* possui cerca de 13 GB de CPU-RAM e por volta de 15 GB de GPU-RAM em seu plano gratuito. Por sua vez, meu computador local possui limitações que podem ser vistas pela figura 1.

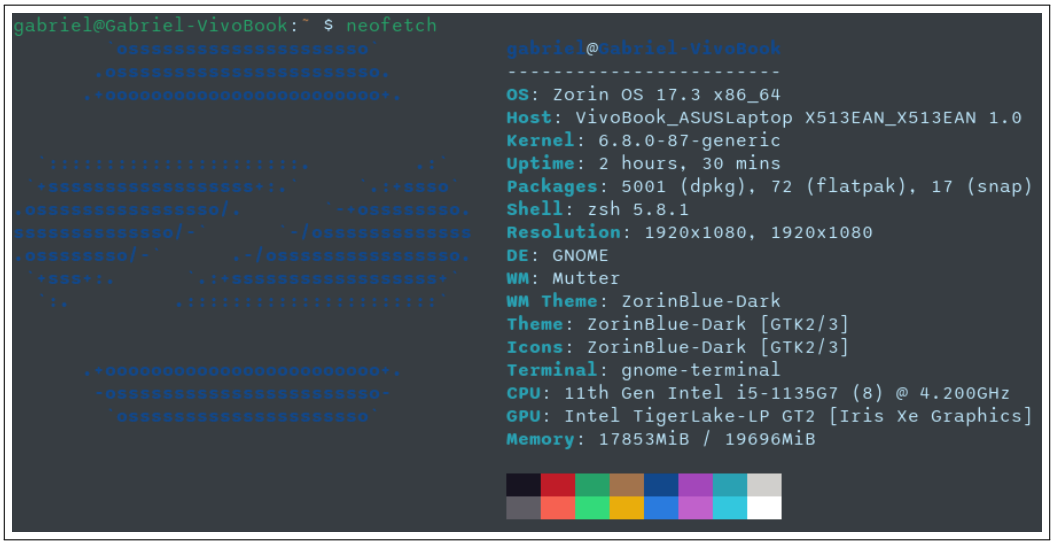
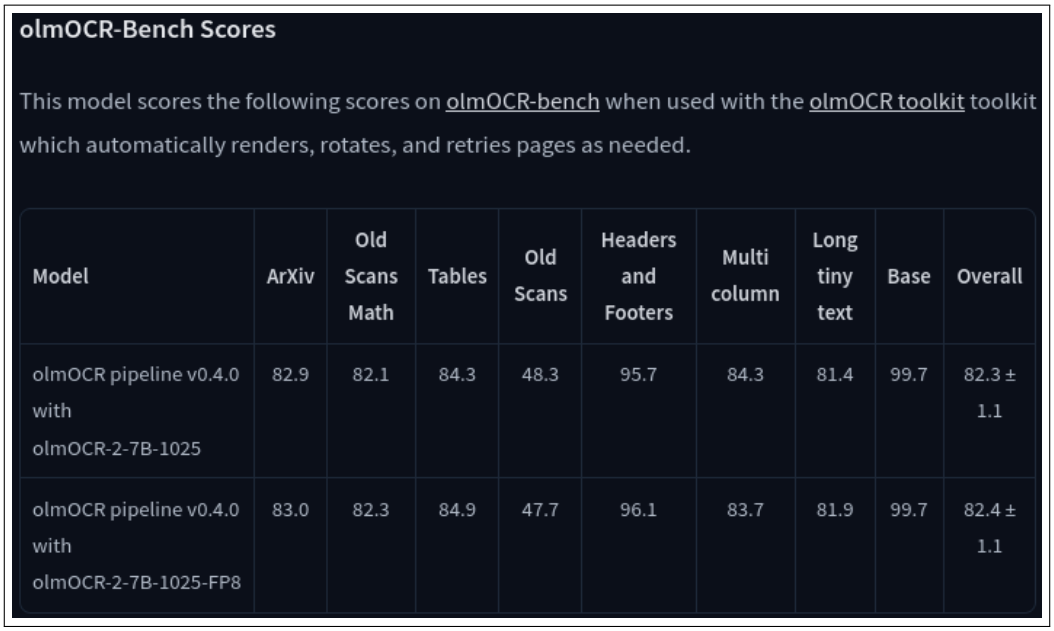


Figura 1: Configuração de *hardware* de computador pessoal

Após a busca inicial de modelos, foram considerados cinco modelos e três variações, sendo duas delas quantizadas para ser possível utilizar em máquinas com menor poder de processamento e outra variação sendo um modelo que já foi utilizada a técnica de *fine-tuning*. A lista completa dos modelos, em ordem alfabética, considerados inicialmente pode ser conferida abaixo:

- allenai/olmOCR-2-7B-1025-FP8: `https://huggingface.co/allenai/olmOCR-2-7B-1025-FP8`
- deepseek-ai/DeepSeek-OCR: `https://huggingface.co/deepseek-ai/DeepSeek-OCR`
- Jalea96/DeepSeek-OCR-bnb-4bit-NF4: `https://huggingface.co/Jalea96/DeepSeek-OCR-bnb-4bit-NF4` - modelo quantizado
- lightonai/LightOnOCR-1B-1025: `https://huggingface.co/lightonai/LightOnOCR-1B-1025`
- microsoft/Florence-2-large: `https://huggingface.co/microsoft/Florence-2-large`
- microsoft/Florence-2-large-ft: `https://huggingface.co/microsoft/Florence-2-large-ft` - modelo que já passou por técnica de *fine-tuning*
- PaddlePaddle/PaddleOCR-VL: `https://huggingface.co/PaddlePaddle/PaddleOCR-VL`
- winninghealth/olmOCR-2-7B-1025-INT4: `https://huggingface.co/winninghealth/olmOCR-2-7B-1025-INT4` - modelo quantizado

Estes modelos foram considerados por conta de dados de *benchmarks* públicos e por dados disponibilizados pela própria empresa, como representados nas figuras 2 e 3.

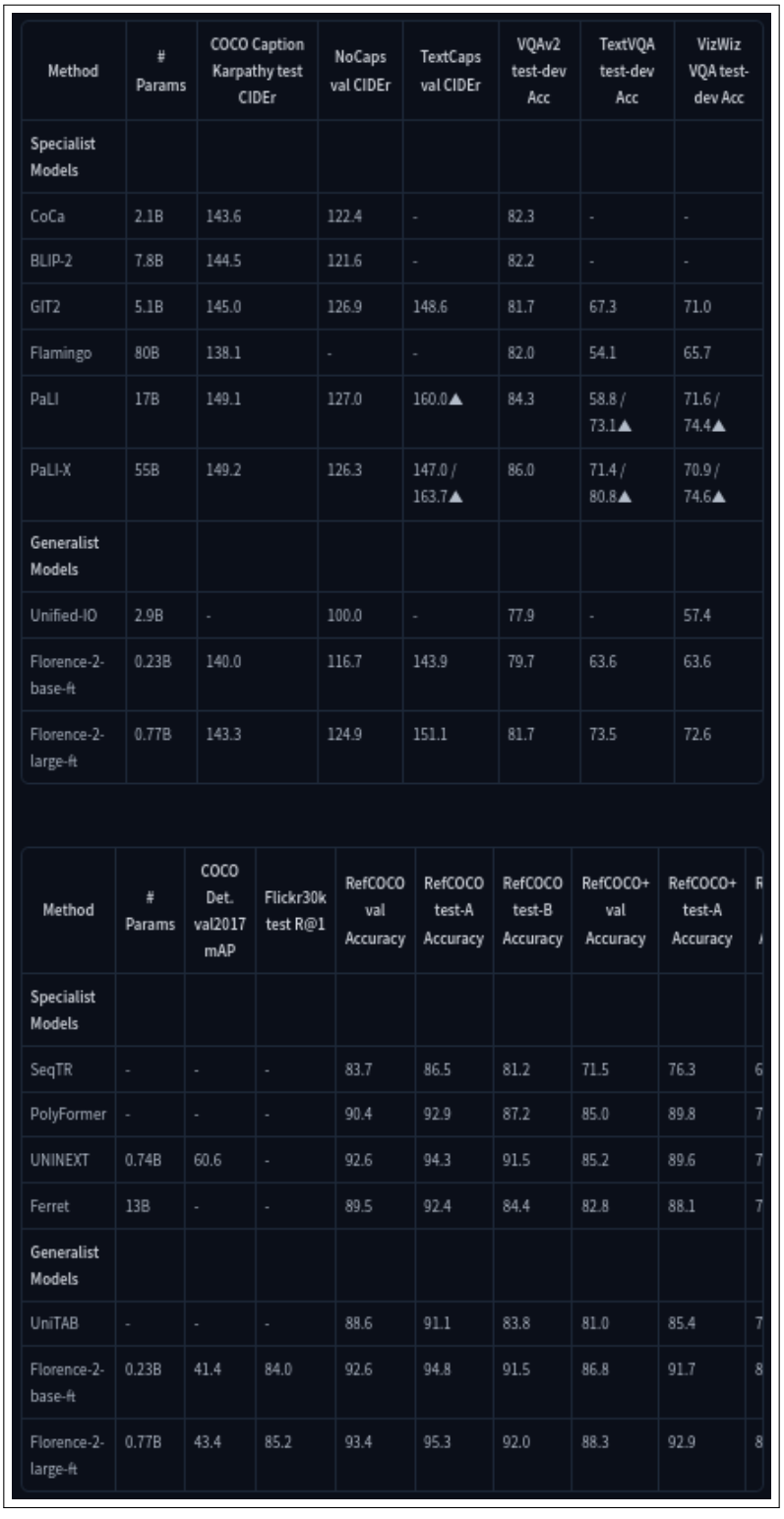


olmOCR-Bench Scores

This model scores the following scores on olmOCR-bench when used with the olmOCR toolkit toolkit which automatically renders, rotates, and retries pages as needed.

Model	ArXiv	Old Scans Math	Tables	Old Scans	Headers and Footers	Multi column	Long tiny text	Base	Overall
olmOCR pipeline v0.4.0 with olmOCR-2-7B-1025	82.9	82.1	84.3	48.3	95.7	84.3	81.4	99.7	82.3 ± 1.1
olmOCR pipeline v0.4.0 with olmOCR-2-7B-1025-FP8	83.0	82.3	84.9	47.7	96.1	83.7	81.9	99.7	82.4 ± 1.1

Figura 2: Dados disponibilizados pela organização AllenAI em sua página no *site Hugging Face*



Method	# Params	COCO Caption Karpathy test CIDEr	NoCaps val CIDEr	TextCaps val CIDEr	VQAv2 test-dev Acc	TextVQA test-dev Acc	VizWiz VQA test-dev Acc
Specialist Models							
CoCa	2.1B	143.6	122.4	-	82.3	-	-
BLIP-2	7.8B	144.5	121.6	-	82.2	-	-
GIT2	5.1B	145.0	126.9	148.6	81.7	67.3	71.0
Flamingo	80B	138.1	-	-	82.0	54.1	65.7
PaLI	17B	149.1	127.0	160.0▲	84.3	58.8 / 73.1▲	71.6 / 74.4▲
PaLI-X	55B	149.2	126.3	147.0 / 163.7▲	86.0	71.4 / 80.8▲	70.9 / 74.6▲
Generalist Models							
Unified-IO	2.9B	-	100.0	-	77.9	-	57.4
Florence-2-base-ft	0.23B	140.0	116.7	143.9	79.7	63.6	63.6
Florence-2-large-ft	0.77B	143.3	124.9	151.1	81.7	73.5	72.6

Method	# Params	COCO Det. val2017 mAP	Flickr30k test R@1	RefCOCO val Accuracy	RefCOCO test-A Accuracy	RefCOCO test-B Accuracy	RefCOCO+ val Accuracy	RefCOCO+ test-A Accuracy	R
Specialist Models									
SeqTR	-	-	-	83.7	86.5	81.2	71.5	76.3	6
PolyFormer	-	-	-	90.4	92.9	87.2	85.0	89.8	7
UNINEXT	0.74B	60.6	-	92.6	94.3	91.5	85.2	89.6	7
Ferret	13B	-	-	89.5	92.4	84.4	82.8	88.1	7
Generalist Models									
UniTAB	-	-	-	88.6	91.1	83.8	81.0	85.4	7
Florence-2-base-ft	0.23B	41.4	84.0	92.6	94.8	91.5	86.8	91.7	8
Florence-2-large-ft	0.77B	43.4	85.2	93.4	95.3	92.0	88.3	92.9	8

Figura 3: Dados disponibilizados pela organização Microsoft em sua página *site Hugging Face*

Além do uso destes dados, também foram consideradas as execuções de demonstrações públicas, especialmente pelo modelo *olmOCR-2-7B-1025-FP8* que teve um desempenho surpreendente, como pode ser visto pelas figuras 4 e 5.

Figura 4: Demonstração pública utilizando a CNH disponibilizada como documento de teste

Figura 5: Demonstração pública utilizando a fatura de energia disponibilizada como documento de teste

3 Técnicas e principais arquiteturas

Primeiramente, foram feitas pesquisas e considerações acerca das principais arquiteturas a serem utilizadas pela prova de conceito, POC. Durante as pesquisas e avaliações, foram consideradas as seguintes abordagens:

- *LLM* puro com *OCR* tradicional: esta abordagem utilizaria um *LLM* puramente textual de alto desempenho com o texto linearizado fornecido por um motor *OCR*, porém neste caso poderiam ter altas perdas de informação espacial. *LLMs* quando

alimentados com textos simpes, não conseguem inferir relações de layout, hierarquia e posição, que são de extrema importância para documentos visualmente complexos. Assim, mesmo que o *LLM* utilizado seja excelente em raciocínio textual, a fundação de dados poderia ser um problema para esta abordagem.

- *Pipeline* modular: esta abordagem utilizaria um *pipeline* completamente modular que combinam ferramentas especializadas, como por exemplo técnicas de *OCR*, *LayoutLM* e *NLU*. Provavelmente esta técnica teria extrema precisão porém por conta da complexidade arquitetural seria inviável de ser feita uma prova de conceito utilizando esta abordagem. Além disso, é provável que a integração de múltiplos módulos aumentariam a latência e o custo de manutenção, afastando-se do objetivo inicial de simplificar e acelerar a *API*.

- *VLM end-to-end* otimizado: esta técnica propõe o uso de um modelo de visão e linguagem multilíngue capaz de ler imagens do documento e gerar a saída já estruturada em um único passo. Esta abordagem resolveria o problema de complexidade visual e eliminaria a latência apresentada da arquitetura. Nesta etapa, foram considerados modelos eficientes em recursos, como é o caso do modelo *PaddleOCR-VL-0.9B* e modelos que possuem larga documentação acerca de técnicas de *fine-tuning*, como é o caso da família de modelos *LLaVA* e *Donut*.

- *SLM*/*LLM* adaptado com estruturação forçada: nesta abordagem, independentemente da escolha do modelo base, sendo um *LLM* ou *VLM*, seriam utilizadas métodos para a estruturação forçada ao *JSON*, como por exemplo utilizando a biblioteca *Pydantic*. Além disso, poderiam ser utilizadas ferramentas como *Outlines* e *Instructor* que poderiam utilizar o *schema Pydantic* para impor restrições na geração do modelo em níveis de *token*. A integração por meio da *PydanticOutputParser* com *frameworks*, como por exemplo o *LangChain*, é considerado um método prático e funcional.

- *RAG* multimodal: de forma geral, o *RAG* tradicional possui diversas limitações com dados estruturados e complexo. Desta forma, esta abordagem indica a utilização de *RAG* multimodal, utilizando *frameworks* como *LangChain* ou *LlamaIndex*, integrados com parsers, como é o caso da biblioteca *Unstructured*. Isso criaria um *pipeline* capaz de lidar com os documentos extensos e que possuem textos e estruturas consideradas complexas.

Em um segundo momento, foram consideradas estratégias de otimização para buscar reduzir o fator de custo pela latência dos modelos. Em um primeiro momento foram considerados *frameworks* de inferência diretamente em Python, como é o caso do conhecido *vLLM* e do *TensorRT-LLM*. O *framework vLLM* é amplamente recomendado devido à sua capacidade de fornecer altos volumes de dados em baixa latência, já o *TensorRT-LLM* é uma alternativa para infraestruturas que priorizam a máxima eficiência em *GPUs* mais recentes e com maior poder computacional, estando profundamente integradas ao ecossistema *NVIDIA*. Apesar disso, para esta prova de conceito, foi utilizada a biblioteca *Transformers* para buscar compreener as limitações de cada um dos modelos integrados aos ambientes de testes.

Por fim, a última técnica de otimização para permitir o uso dos modelos em *hardware* de custo-benefício foi a quantização dos modelos. Atualmente, existem diversas pesquisas acerca deste tema e diversas implementações, conforme os itens abaixo:

- *Activation-Aware Weigth Quantization* (*AWQ*): esta técnica realizada no pós-treinamento do modelo reduz a pegada de VRAM de forma drástica buscando manter a alta precisão ao preservar os pesos mais "importantes"do modelo
- Uso de FP8 e FP16: o uso de *floating-points* de menor precisão reduz os requisitos de memória pela metade tendo em vista a redução de *bits* utilizados para representar os números a serem utilizados pelo modelo
- Uso de bibliotecas específicas: o uso de bibliotecas, como a *bitsandbytes*, busca realizar a quantização do modelo para que ele seja armazenado em menor espaço na memória do ambiente de execução

4 Experimentos e resultados

Após definir os principais modelos a serem testados, foram escritos arquivos no formato *.ipynb* para poderem ser testados tanto localmente quanto no ambiente do *Google Colab*. Inicialmente, seriam feitos arquivos separados para cada um para que pudessem ser feitos comparativos entre os modelos.

Infelizmente, alguns dos modelos não puderam ser experimentados por razões distintas, sendos elas:

- allenai/olmOCR-2-7B-1025-FP8: o modelo mais promissor e com melhores resultados até então, não pôde ser experimentado em nenhum dos ambientes. Em meu computador pessoal, mesmo possuindo 20 GB de memória RAM, não foi possível alocar todo o modelo e, assim como em meu computador, não foi possível utilizar o modelo no ambiente virtual por falta de memória na placa de vídeo, NVIDIA T4, disponibilizada pela *Google*
- deepseek-ai/DeepSeek-OCR: assim como o modelo anterior, não foi possível realizar experimentos com este modelo por conta da falta de memória nos ambientes de testes, porém neste caso foi possível realizar testes com seu modelo respectivo quantizado
- lightonai/LightOnOCR-1B-1025: diferentemente dos modelos anteriores, este modelo pôde ser carregado porém por não possuir suporte à biblioteca *Transformers*, não foi possível utilizá-lo

Deste modo, foram realizados testes simplificados com os modelos restantes, de forma que todos os experimentos basearam-se em carregar o modelo e realizar a inferência de *OCR* no primeiro e mais simples caso de teste. Todos os códigos desta etapa podem ser vistos na pasta *experiments* do repositório referente à este desafio técnico.

Por fim, como última parte desta etapa, foi feita uma tabela comparativa entre os modelos escolhidos, de forma que os valores estão em escala positiva, ou seja, quanto

maior, melhor. As métricas escolhidas e os valores atribuídos podem ser vistos na tabela 1. Todos os valores foram obtidos experimentalmente por meio da pesquisa e dos códigos implementados para este dossiê.

Tabela 1: Requisitos para a PCI

Métricas	**DeepSeek-OCR-bnb-4bit-NF4**	**Florence-2-large**	**PaddleOCR-VL**	**olmOCR-2-7B-1025-INT4**
Precisão	7	6	6	10
Layout	7	5	6	9
Multilíngue	6	8	6	9
Tempo de inferência	4	10	10	4
Consumo de VRAM/RAM	4	9	9	8
Facilidade de uso	8	9	2	7
Compatibilidade	8	10	2	8
Fine-tuning	8	9	2	9

5 Análise de viabilidade de implementação

Para a viabilidade de implementação das provas de conceito, foram realizados dois códigos distintos. Em um primeiro momento, foi feita uma prova de conceito para validar o uso de bibliotecas como a *FastAPI*, onde neste momento foi utilizado o modelo disponibilizado pela Microsoft, Florence-2-large. Vale ressaltar que nesta prova de conceito, os dados fornecidos pelo *JSON* são dados fictícios, usados apenas para validar as interfaces implementadas, apesar de também mostrar ao usuário o retorno fornecido pelo modelo.

Por sua vez, em uma segunda implementação, utilizando um arquivo *Jupyter Notebook* e o ambiente fornecido pela *Google* foi feita uma segunda prova de conceito, onde o modelo recebeu os três arquivos fornecidos para o desafio e diferentes abordagens foram feitas de acordo o tipo de arquivo recebido.

Deste modo, a partir da segunda prova de conceito, entende-se que o melhor caminho para possivelmente substituir a atual *API* seria o uso do modelo quantizado *olmOCR-2-7B-1025-INT4*. O modelo, durante a prova de conceito não passou a utilizar mais do que 10 GB de RAM, tanto da CPU quanto da GPU. Este fato, levanta um item extremamente importante para o dimensionamento da possível máquina a alocar a nova arquitetura da *API*, afinal, o computador poderá ter especificações próximas àquelas utilizadas no ambiente de testes. Para esta etapa, foram encontradas duas soluções interessantes para a locação de máquinas com alto poder de processamento, sendo elas:

- *Lambda Labs*: empresa especializada em computação para inteligência artificial, que disponibiliza máquinas físicas quanto instâncias em nuvem com GPUs NVIDIA, sendo utilizada amplamente por pesquisadores e desenvolvedores.

- *RunPod*: plataforma de computação distribuída que oferece acesso remoto à GPUs, porém com foco em flexibilidade e custo-benfício, especialmente pelo fato de poder pausar a máquina alugada para reduzir custos.

Assim, considerando as informações obtidas pela prova de conceito e pela descrição anterior, foram feitas simulações de locação de máquinas pelo serviço *RunPod*, sendo que ambas foram selecionadas com o serviço de criptografia em nuvem e os valores finais podem ser vistos pelas figuras 6 e 7. Vale ressaltar que poderiam ser feitos estudos acerca do melhor horário para manter o servidor ativo para reduzir ainda mais os custos, especialmente caso os usuários finais sejam de empresas que só possam utilizar os serviços em horários pré-determinados.

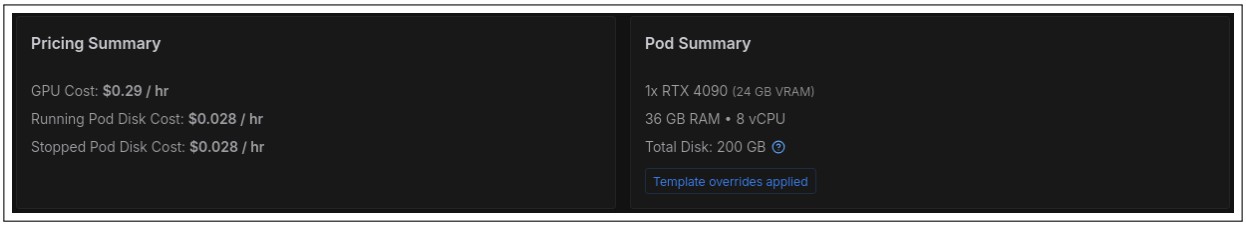


Figura 6: Simulação de locação de computador com placa de vídeo RTX 4090

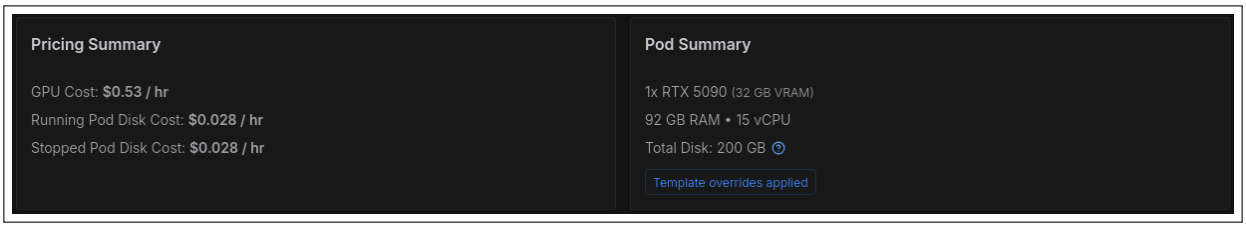


Figura 7: Simulação de locação de computador com placa de vídeo RTX 5090

6 Conclusões e recomendação final

A partir das informações fornecidas pelo documento introdutório e pela documentação fornecida, entende-se que a solução para os problemas de latência, custo e complexidade da *API* de extração de documentos reside em uma mudança da arquitetura atual. A arquitetura de duas etapas baseada em LLM textual deve ser substituída por um *VLM end-to-end*, como é o caso do modelo, já quantizado, *winninghealth/olmOCR-2-7B-1025-INT4*, especialmente considerando a saída recebida pelos casos de teste, conforme representado abaixo para os casos de teste com a CNH e a fatura de energia:

```
CNH: {
  "document_type": "Brazilian National ID Card",
```

```
  "document_number": "00123456789",
  "name": "Lince da Silva",
  "date_of_birth": "02/05/2018",
  "place_of_birth": "RIO DE JANEIRO",
  "gender": "M",
  "mother_name": "Maria de Silva",
  "father_name": "Jose da Silva",
  "issuing_authority": "DEPARTAMENTO NACIONAL DE TRANSPORTES - CARTOES
     NACIONAIS DE IDENTIDADE",
  "issuing_date": "22/10/2019",
  "place_of_issue": "BRASILIA - DISTRITO FEDERAL, DF",
  "qr_code": "61147258369 0773799567"
}

Fatura de energia: {
  "cliente": {
    "nome": "Maria Jose da Silva",
    "cpf": "123.456.789-10"
  },
  "endereco": {
    "rua": "Joao Fernandes Vieira 175",
    "bairro": "Boa Vista",
    "cep": "50680-150"
  },
  "conta_contrato": "1234567890",
  "mes": "06/2014",
  "data_vencimento": "10/07/2014",
  "data_prevista_programa_leitura": "25/07/2014",
  "total_a_pagar": "63,72",
  "consumo": {
    "consumo_alvo": "160.000000",
    "preco": "0,39825738",
    "valor": "63,72"
  },
  "historico_consumo": [
    {
      "medidor": "1234567890",
      "tipo_funcionamento": "CAT",
      "data_anterior_lectura": "27/06/2014",
      "data_atual_lectura": "24/07/2014",
      "consumo_alvo": "160.000000",
      "consumo": "1.50850000"
    }
  ],
  "informacoes_de_trIBUTOS": [
```

```
    {
      "base_calculo": "0,372",
      "percentual": "17,00",
      "valor_do_calculo": "62,72",
      "tributo": "ICMS"
    },
    {
      "base_calculo": "0,372",
      "percentual": "0,99",
      "valor_do_calculo": "0,37",
      "tributo": "PIS"
    },
    {
      "base_calculo": "0,372",
      "percentual": "4,54",
      "valor_do_calculo": "17,00",
      "tributo": "COFINS"
    }
  ],
  "composicao_consumo": [
    {
      "consumo": "20,28",
      "porcentagem": "41,46",
      "tributo": "GERACAO DE ENERGIA"
    },
    {
      "consumo": "1,72",
      "porcentagem": "2,75",
      "tributo": "TRANSMISSAO"
    },
    {
      "consumo": "19,07",
      "porcentagem": "39,93",
      "tributo": "DISTRIBUICAO (Carga)"
    },
    {
      "consumo": "2,22",
      "porcentagem": "4,22",
      "tributo": "DISTRIBUICAO (Setor)"
    },
    {
      "consumo": "14,05",
      "porcentagem": "25,22",
      "tributo": "Tributos"
    },
```

```
    {
      "consumo": "63,72",
      "porcentagem": "100",
      "tributo": "TOTAL"
    }
  ]
}
```

Deste modo, o processo é single-pass, sem a necessidade de duas inferências e após a aplicação de técnicas de *fine-tuning*, o modelo irá se adaptar de forma mais eficiente ainda às regras específicas de extração e formatação de documentos brasileiros, como é o caso da carteira nacional de habilitação e de faturas energéticas, garantindo melhores métricas .

Por fim, a nova abordagem irá resolver a complexidade visual, reduzir a latência por ter apenas uma única inferência e permitir que a empresa possua controle total sobre o custo operacional por meio da auto-hospedagem do modelo de código aberto.

7 Referências

Como referências desta pesquisa, abaixo estão os principais *links* utilizados e também os títulos dos artigos lidos durante as pesquisas.

Links úteis:

- arXiv.org: `https://arxiv.org/`
- Demonstração pública do modelo *allenai/olmOCR-2-7B-1025-FP8*: `https://olmocr.allenai.org/`
- *Hugging Face*: `https://huggingface.co/`
- *Kaggle*: `https://www.kaggle.com/`

Artigos lidos:

- DeepSeek-OCR: Contexts Optical Compression
- LongDocURL: a Comprehensive Multimodal Long Document Benchmark Integrating Understanding, Reasoning and Locating
- OCR-free Document Understanding Transformer
- OCR-Reasoning Benchmark: Unveiling the True Capabilities of MLLMs in Complex Text-Rich Image Reasoning
- olmOCR-2-Unit-Test-Rewards-for-Document-OCR
- OmniVinci: Enhancing Architecture and Data for Omni-Modal Understanding LLM

- PaddleOCR-VL: Boosting Multilingual Document Parsing via a 0.9B Ultra-Compact Vision-Language Model