Leveraging LLMs for Automated Generation and Validation of Financial Descriptions for Lithuanian Companies

Antanas Baltrušaitis

PyCon 2025

This topic is for...

- Broke devs, for whom LLM AI API cost is a burden
- Devs who can‘t use public LLM AI APIs due to privacy or other reasons
- Devs who interested in local LLM deployments
- Devs who interested in local AI translation
- Devs who are interested in fast local mass generation and translation

This topic is not for...

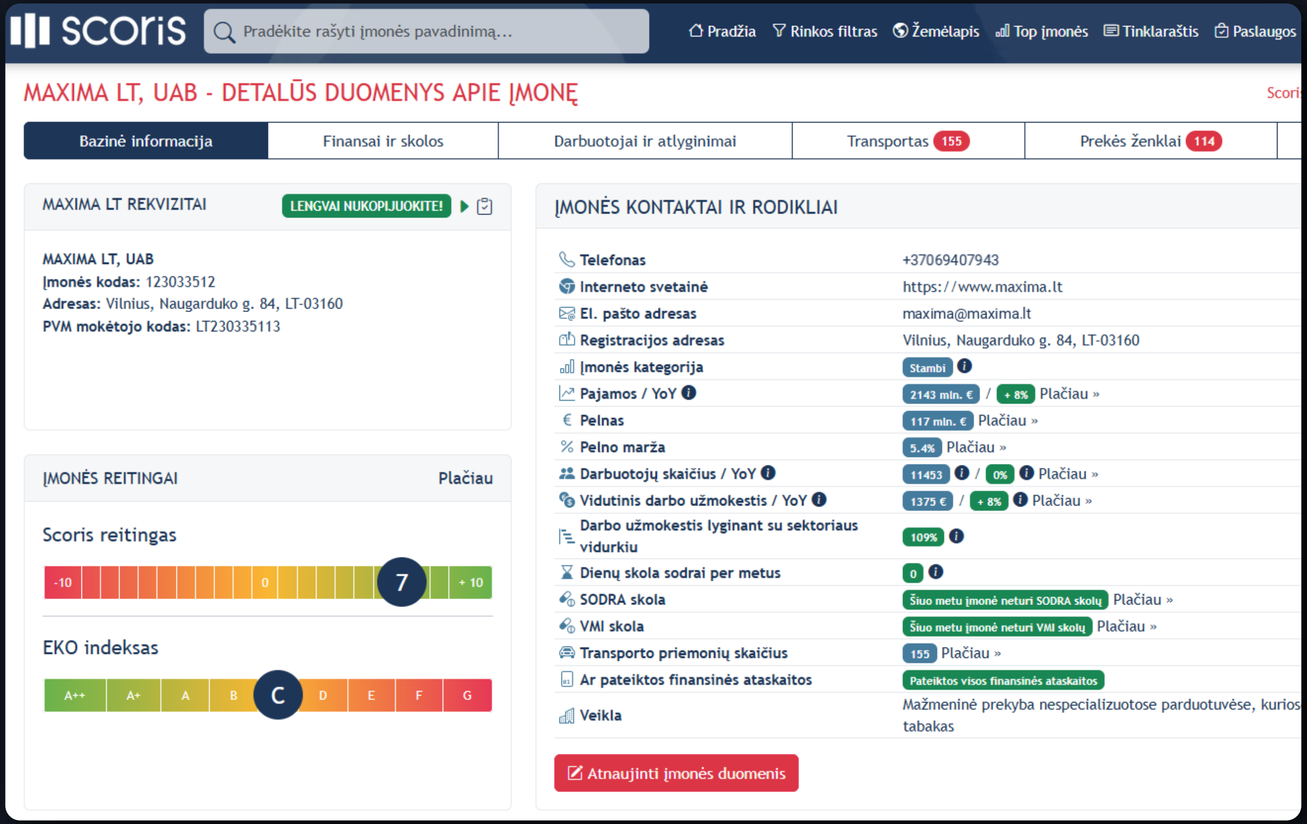
- Devs who are using OpenAI API (or any other) service can afford it and are happy with it

Hi!

Antanas Baltrušaitis

- Founder @ scoris.lt
- Creator @ oriux.lt
- Sn. Analytics engineer @ Beyond Analysis
- Open data and AI enthusiast

scoris

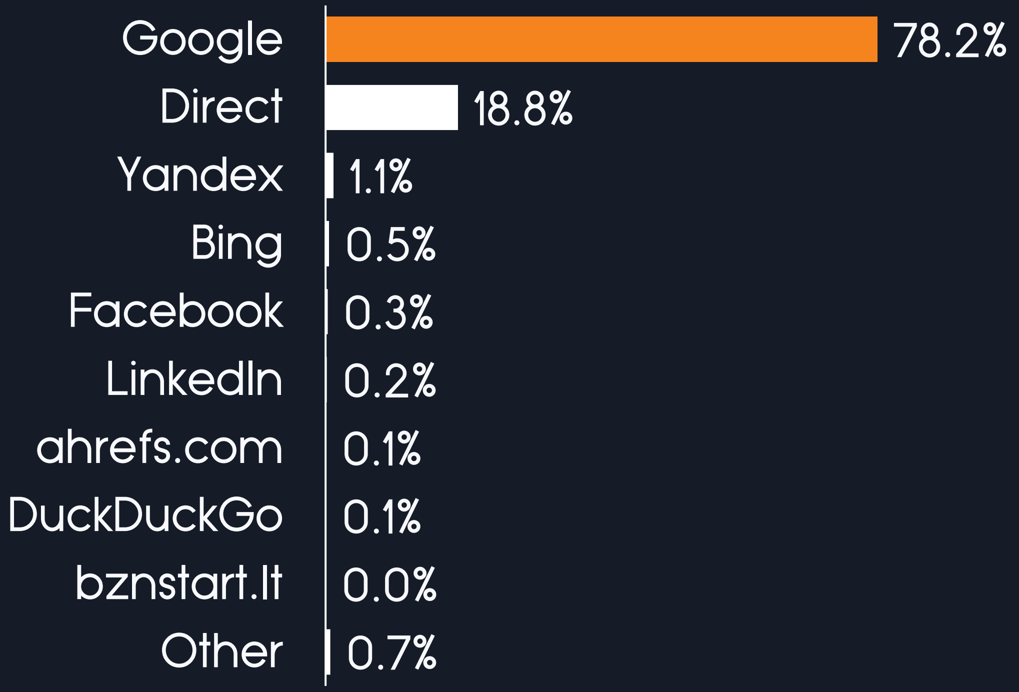


- Comprehensive coverage of all Lithuanian companies (240,000 active and 270,000 inactive)
- Over 100 public data sources, ~300GB database
- Innovative and unique market tools for finding target companies and performing market research ("Market Filter", "Business Map")
- Advanced APIs
- Engaging market intelligence through our Blog and LinkedIn feed and press

The Problem

Google – main traffic source

Scoris traffic sources:



- SEO **goals** in a nutshell:
  - Get indexed
  - Improve ranking
- Google Guidelines to Enhance Ranking:
  - Provide relevant content
    - *Relevant "text"*
  - Offer a great user experience
    - *High page-speed-index ranking*
  - Establish a reputable website
    - *Backlink profile*

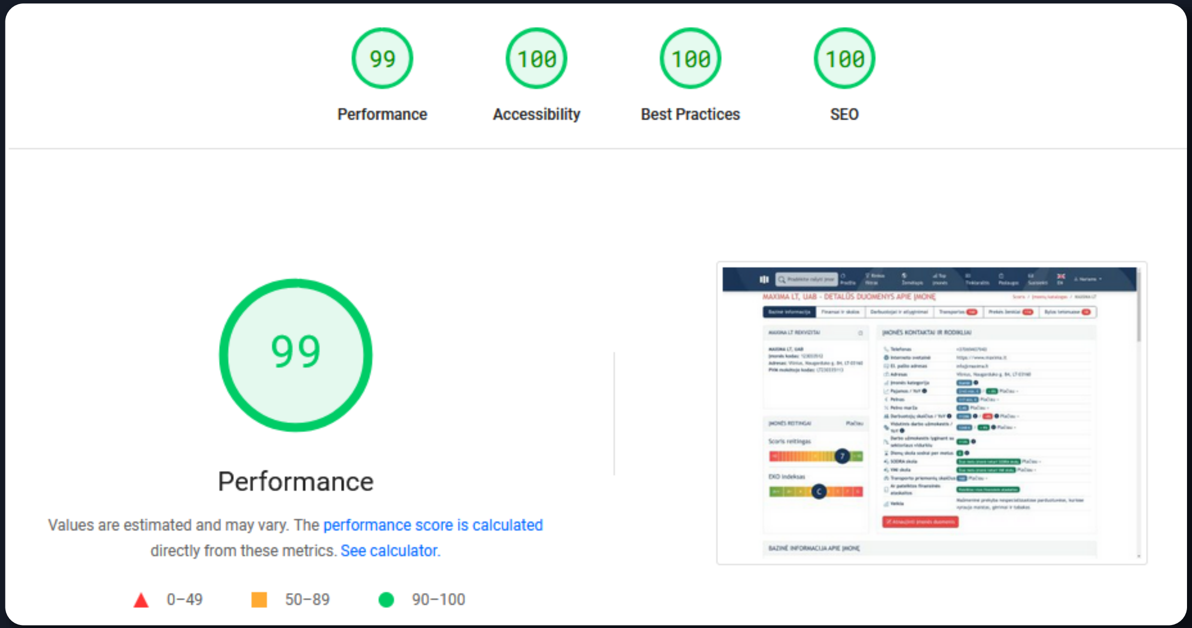
“Hacking” google search

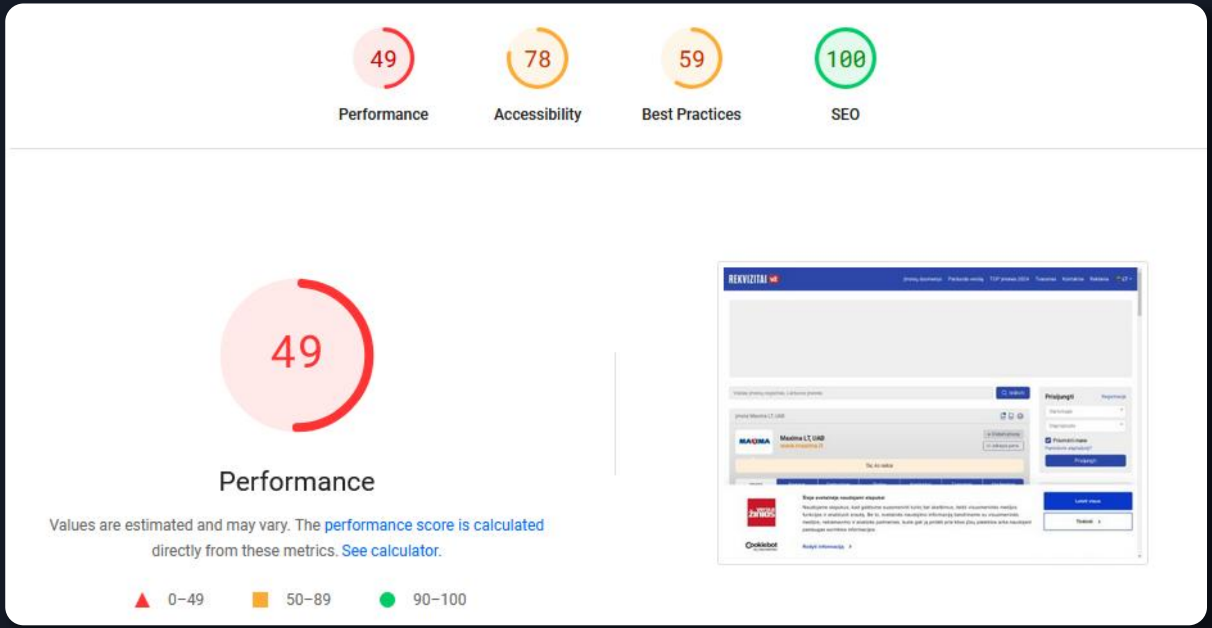
All pages are already indexed by Google, but avg. position was ~12.

All measurable aspects are fine-tuned to perfection...

scoris

Competitor

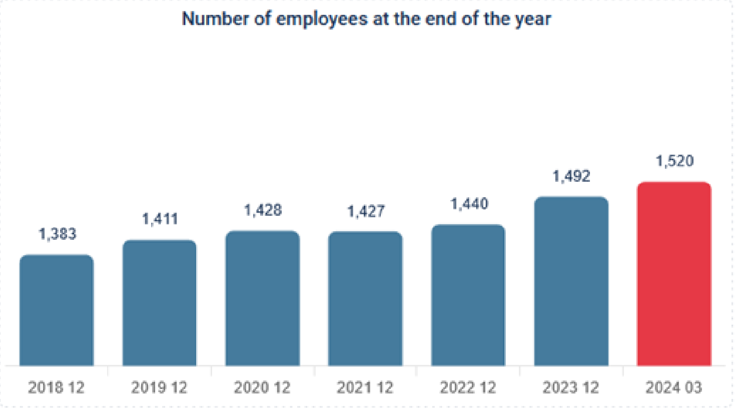




If **quantitative** is perfect, then the problem is **qualitative**...

We have plenty of beautiful data, but almost no text

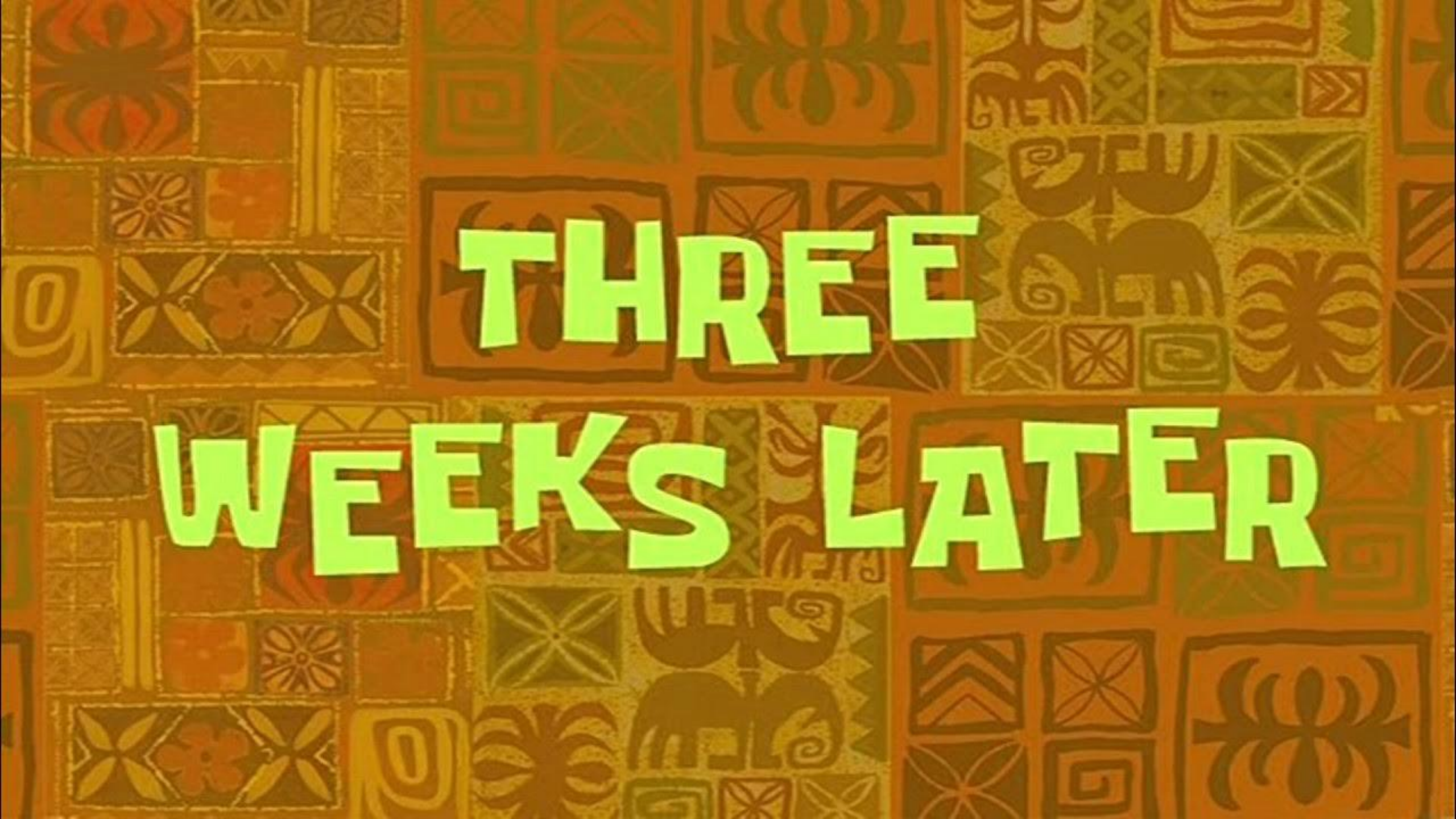
million EUR	2022	2021	2020	2019	2018
Financial data:					
Sales revenue	7,552	4,264	2,426	4,503	4,625
Profit before tax	-162	76	-186	59	18
Net profit	-183				
Personal capital	286	442	337	520	448
Obligations	865				
Fixed assets	323	420	332	303	260
Current assets	827	731	321	604	540
Property of the whole	1,150	1,151	653	907	800
Financial indicators:					
Change in income y/y	+77.1%	+75.7%	-46.1%	-2.7%	+16.0%
ROA	-15.9%	-	-	-	-
ROE	-64.1%	-	-	-	-
Profit margin	-2.4%	-	-	-	-
EBT%	-2.1%	1.8%	-7.7%	1.3%	0.4%
Liabilities/Equity ratio	3.0	-	-	-	-
Income for the employee	5,237,453	2,965,968	1,699,966	3,216,653	3,344,874



Google does not like tabular data, and it can't process charts…
In a result – this makes our "good data" look like "**less relevant**" to search engines, negatively affecting search rankings.

Let's use local large language model, which will generate financial descriptions in Lithuanian

(as cheap as possible; as fast as possible; decent quality)


THREE
WEEKS LATER

4 seconds per description

Total generation time: 7 days


6 MONTHS
LATER...

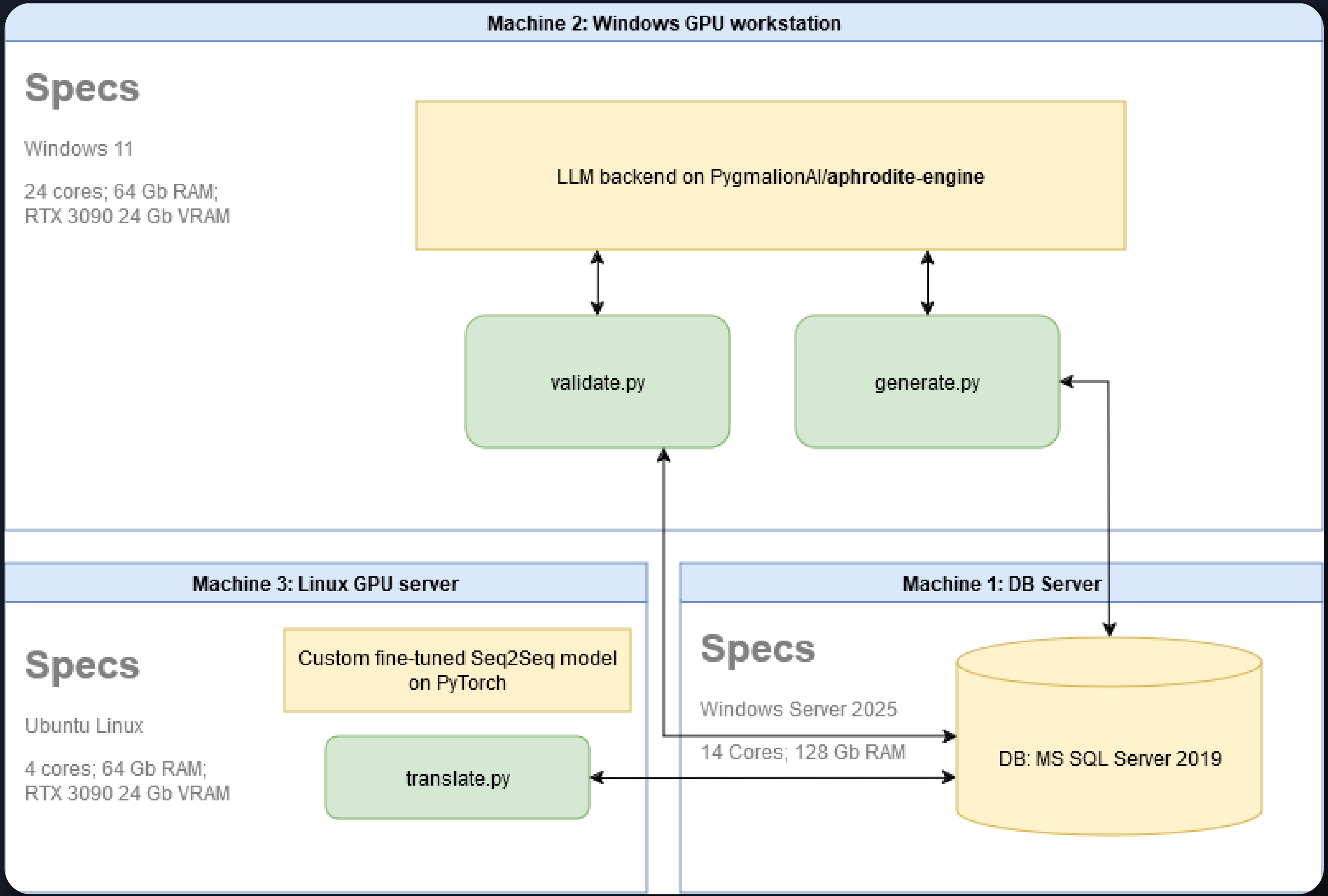
0.6 seconds per description

Total generation time: 1 day!

The Solution

Broke Dev's Guide to AI solutions

	Cloud service providers	DIY on-prem setup
Generate	**Pricing:** Input ~$4; Output ~$15 / 1M tokens	Open source LLM model local deployment
Translate	DeepL **Pricing:** $20 / 1M characters	Open-source Machine Translation model local deployment
	Setup cost: 0 **Output cost:** ~€6000 **Time to market:** short	**Setup cost:** ~€1000 **Output cost:** ~ €100 (electricity) **Time to market:** long



Solution

generate.py:
Fetches raw data and generates English description

validate.py:
Fetches raw data and generated description and validates if it's correct

transalte.py
Translates English description to Lithuanian

Same can be achieved on one machine ☺
Since we had 3 – we have used all 3.

LLM backend: Aphrodite engine

https://github.com/PygmalionAI/aphrodite-engine

Setup:

```
1. pip install -U aphrodite-engine
2. aphrodite run Qwen/Qwen2.5-7B-Instruct-GPTQ-Int4
```

```
curl -X POST http://localhost:2242/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "Qwen/Qwen2.5-7B-Instruct-GPTQ-Int4",
  "messages": [
    {
      "role": "system",
      "content": "You are helpful assistant"
    },
    {
      "role": "user",
      "content": "YOUR_PROMPT_HERE"
    }
  ],
  "mode": "instruct",
  "seed": "$RANDOM",
  "stream": false
}'
```

This will open (OpenAI) API POST endpoint to send chat completions requests to. This allows continuous batching and concurrent generation.

Models of choice:

- Meta-Llama-3-8B-Instruct-6.0bpw-h6-exl2
- Qwen2.5-7B-Instruct-GPTQ-Int4

!!! Generation throughput is ~1000 tokens/s on single RTX 3090 GPU !!!

Generate.py / validate.py
1. Setup & Configuration

```
import aiohttp, asyncio, pandas as pd, pyodbc
from datetime import datetime

batch_size = 30
semaphore = asyncio.Semaphore(batch_size)
df_columns = ['company_code', 'prompt', 'seed', 'model', 'financial_data',
              'generated_description_en', 'description_type', 'tokens']
results_df = pd.DataFrame(columns=df_columns)
```

- **Async-focused imports (aiohttp, asyncio).** Allowing multiple API calls to be processed simultaneously without blocking
- **Data processing tools** (pandas, pyodbc)
- **Concurrency control with semaphore**. To limit concurrent API calls to 30 at a time, preventing server overload while maintaining efficient throughput.
- **Data structure setup with pandas**

Generate.py / validate.py

2. API Integration

```python
async def generate(session, prompt_text, retries=3):
    url = "http://localhost:2242/v1/chat/completions"
    data = {
        "model": "Qwen/Qwen2.5-7B-Instruct-GPTQ-Int4",
        "messages": [
            {"role": "system", "content": "You are a professional business writer"},
            {"role": "user", "content": prompt_text}
        ],
        "seed": random.randint(0, 1000000000),
        "stream": False
    }
    async with session.post(url, json=data, ssl=False) as response:
        response_json = await response.json()
        return response_json['choices'][0]['message']['content'], response_json
```

- Async HTTP requests
- LLM API integration
- Error handling and retries
- Structured prompt engineering
- Random seed for different outcomes

Generate.py / validate.py
3. Data Processing Pipeline

```python
async def process_row(session, row, semaphore):
    async with semaphore:
        company_code, financial_data = row[0], row[1]
        prompt_text = (f"Generate a detailed financial description for the company using the provided data only.\n"
                       f"Important! You must use this data only:\n{clean_data}\n"
                       f"Requirements:\n"
                       f"1. Aim for 300-400 words description.\n"
                       f"2. Ensure all figures are correct.\n"
                       f"3. Use simple sentences and words.\n"
                       f"4. Refer to the entity as 'the Company.'\n"
                       f"5. Do not include any notes, explanations, or assumptions.\n"
                       f"6. Do not include summary facts or footnotes.\n"
                       f"7. It should be ready for publishing as-is copy-paste, so do not use any placeholders, introductions etc.\n"
                       f"8. Description must be SEO friendly with main keywords: financial data, finalcial reports, revenue, profit, finance\n"
                       f"\n")

        description, response = await generate(session, prompt_text)

        new_row = {
            'company_code': company_code,
            'financial_data': financial_data,
            'generated_description_en': description,
            'load_date_time': datetime.now()
        }
        global results_df
        results_df = results_df._append(new_row, ignore_index=True)
```

- Prompt construction
- Concurrent data processing
- Semaphore for rate limiting
- Data transformation pipeline
- DataFrame management

Generate.py / validate.py
4. Orchestration & Database Integration

```python
async def main():
    conn = pyodbc.connect(conn_str)
    query = """SELECT company_code, financial_data
               FROM temp_finance_data
               WHERE NOT EXISTS (
                   SELECT 1 FROM llm_finance_descriptions
                   WHERE company_code = temp_finance_data.company_code
               )"""
    rows = conn.execute(query).fetchall()

    async with aiohttp.ClientSession() as session:
        tasks = [process_row(session, row, semaphore) for row in rows]
        for i in range(0, len(tasks), batch_size):
            batch = tasks[i:i + batch_size]
            await asyncio.gather(*batch)
            await insert_to_db(results_df)

if __name__ == "__main__":
    asyncio.run(main())
```

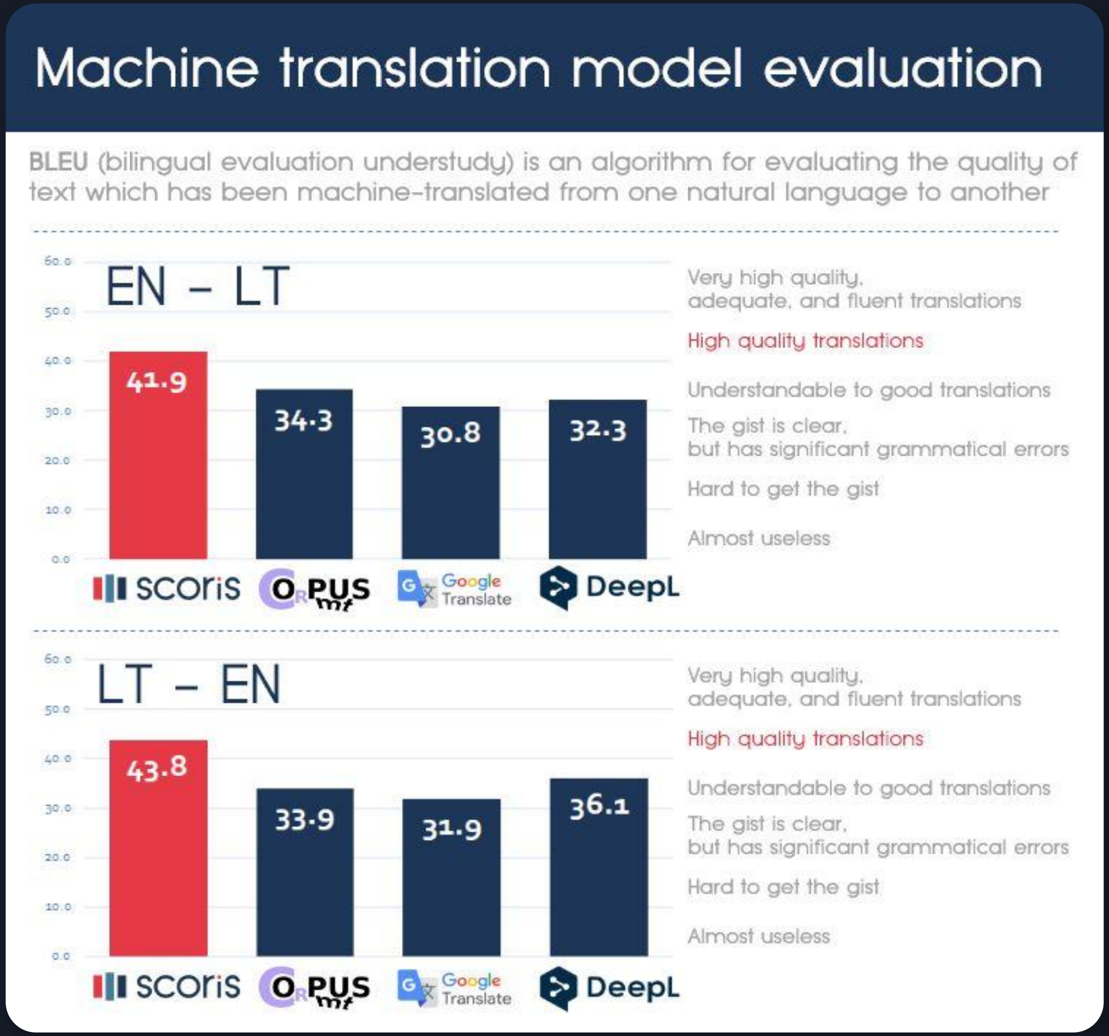
- Async orchestration
- Batch processing
- SQL integration
- Main execution flow

“Validate” prompt

```
prompt_text = (
"You validate facts in the text and label text 'CORRECT' or 'INCORRECT'.\n"
"Instruction:\n"
"1. Validate the text against the provided data set.\n"
"2. Respond with a single-word label: 'Correct' or 'Incorrect'.\n"
"   - 'Correct' if the text matches or is approximately equal to the provided data (rounded values are acceptable).\n"
"   - 'Incorrect' if the text does not match the provided data.\n\n"
"# Text for Validation:\n"
f"\"{row.generated_description_en}\"\n\n"
"# Data for Validation:\n"
f"\"{clean_data}\"\n"
f"NO reasoning. Single word response only.\n"
f"Pay attention to terms growth/decline when comparing number dynamics over time.\n"
f"Your single word ('Correct' or 'Incorrect') assessment is: \n"
)
```

Translate English description to Lithuanian: custom fine-tuned model

- Original Opus MT performed quite well but was not perfect for our use-case.
- We have performed translation model fine-tuning based on public data (5.4 million sentence pairs, equivalent to about 300,000 pages of text) and Custom ~100k sentence pairs tailored for Scoris use-case.
- The Final results were astonishing! Models even surpassed translations of Deepl and Google!
- Models and data-set are published on Huggingface:
  - https://huggingface.co/scoris/scoris-mt-lt-en
  - https://huggingface.co/scoris/scoris-mt-en-lt
  - https://huggingface.co/datasets/scoris/en-lt-merged-data



Translate.py
1. Model Setup & Configuration

```
import torch
import onnxruntime as ort
from transformers import MarianTokenizer
from optimum.onnxruntime import ORTModelForSeq2SeqLM

provider = "CUDAExecutionProvider"

# ONNX Runtime optimization
sess_options = ort.SessionOptions()
sess_options.graph_optimization_level = ort.GraphOptimizationLevel.ORT_ENABLE_ALL
sess_options.intra_op_num_threads = 8
model_trans = ORTModelForSeq2SeqLM.from_pretrained(
    local_model_path,
    provider=provider,
    session_options=sess_options
)
```

- Converted model to ONNX format
- ONNX runtime for optimized inference
- Performance optimization settings
- MarianMT translation model loading

Translate.py
2. Translation Engine

```
def batch_translate(texts_en, batch_size=128):
    translated_texts = []
    for i in range(0, len(texts_en), batch_size):
        batch_texts = texts_en[i:i+batch_size]
        inputs = tokenizer_trans(batch_texts, return_tensors="pt", padding=True)
        with torch.no_grad():
            translations = model_trans.generate(**inputs, num_beams=1)
        batch_translations = tokenizer_trans.batch_decode(translations,
                                                        skip_special_tokens=True)
        translated_texts.extend(batch_translations)
    return translated_texts
```

- **Batch processing for efficiency:** Instead of processing one text at a time, multiple texts are processed simultaneously in batches
- **No-grad inference**. Using torch.no_grad() disables gradient calculation during model inference. Since we're only doing translation (forward pass) and not training, this saves significant memory
- **Greedy decoding strategy**. Using num_beams=1 implements greedy decoding, where at each step, the model selects the most likely next token

Translate.py
3. Text Processing Pipeline

```
def translate(text_en):
    paragraphs = text_en.split('\n\n')
    generated_description_lt = ""
    for para in paragraphs:
        sentences = re.split(r'(?<!\d)\.(?!\d)', para)
        translated_sentences = batch_translate(sentences)
        for translated_sentence in translated_sentences:
            if not translated_sentence.endswith('.'):
                translated_sentence += '.'
            generated_description_lt += translated_sentence + " "
        generated_description_lt += "\n\n"
    return generated_description_lt.strip()
```

- **Paragraph-level processing**. Text is broken down into paragraphs first.
- **Smart sentence splitting**. Uses regex pattern (?<!/d)/.(?!/d) to split text into sentences while avoiding splits at decimal points or numbered lists (like '3.14' or '1.2.3').
- **Sentence formatting**. Ensures each translated sentence ends with a period and has proper spacing to maintain readability and grammatical correctness in the target language.
- **Text structure preservation**. Maintains the original document's formatting by keeping paragraph breaks and spacing intact during translation, ensuring the output matches the input's visual structure.

Translate.py
4. Service Orchestration

```
def main_loop():
    batch_size = 50
    while True:
        conn = pyodbc.connect(conn_str)
        cursor.execute("SELECT ID, generated_description_en FROM llm_finance_descriptions WHERE generated_description_lt is null")
        rows = cursor.fetchall()

        for row in rows:
            translated_description = translate(row.generated_description_en)
            batch_updates.append((translated_description, row.ID))

            if len(batch_updates) >= batch_size:
                cursor.executemany("UPDATE llm_finance_descriptions SET generated_description_lt = ? WHERE ID = ?", batch_updates)
                conn.commit()

                print(f"Processed {processed_items}/{total_items}, "
                      f"Completion: {processed_items/total_items*100:.2f}%")
```

- Continuous service operation. Service runs in an infinite loop, constantly checking for new untranslated records in the database.
- Batch database updates
- Progress monitoring
- Error handling and recovery

Final result

We have generated financial and staff data descriptions for ~115k companies significantly increasing amount of readable text on the website.

THERMO FISHER SCIENTIFIC BALTICS ĮMONĖS FINANSAI

mln. EUR	2023	2022	2021	2020
Finansiniai duomenys:				
Pardavimo pajamos	821	1,477	1,942	1,264
Pelnas prieš apmokestinimą	393	626	715	473
Grynasis pelnas	349	554	618	405
Nuosavas kapitalas	1,186	803	1,859	1,241
Įsipareigojimai	57	109	395	577
Ilgalaikis turtas	178	186	191	153
Trumpalaikis turtas	1,073	736	2,072	1,668
Turtas viso	1,251	922	2,263	1,820
Finansiniai rodikliai:				
Pajamų pokytis y/y	-44.4%	-23.9%	+53.7%	+187.1%
ROA	27.9%	60.1%	27.3%	22.3%
ROE	29.4%	69.0%	33.2%	32.7%
Pelno marža	42.5%	37.5%	31.8%	32.1%

ĮMONĖS FINANSINĖS PADĖTIES ANALIZĖ

Dėmesio! Šis aprašymas sugeneruotas dirbtinio intelekto. Jeigu manote, kad jis netikslus - **praneškite** mums.

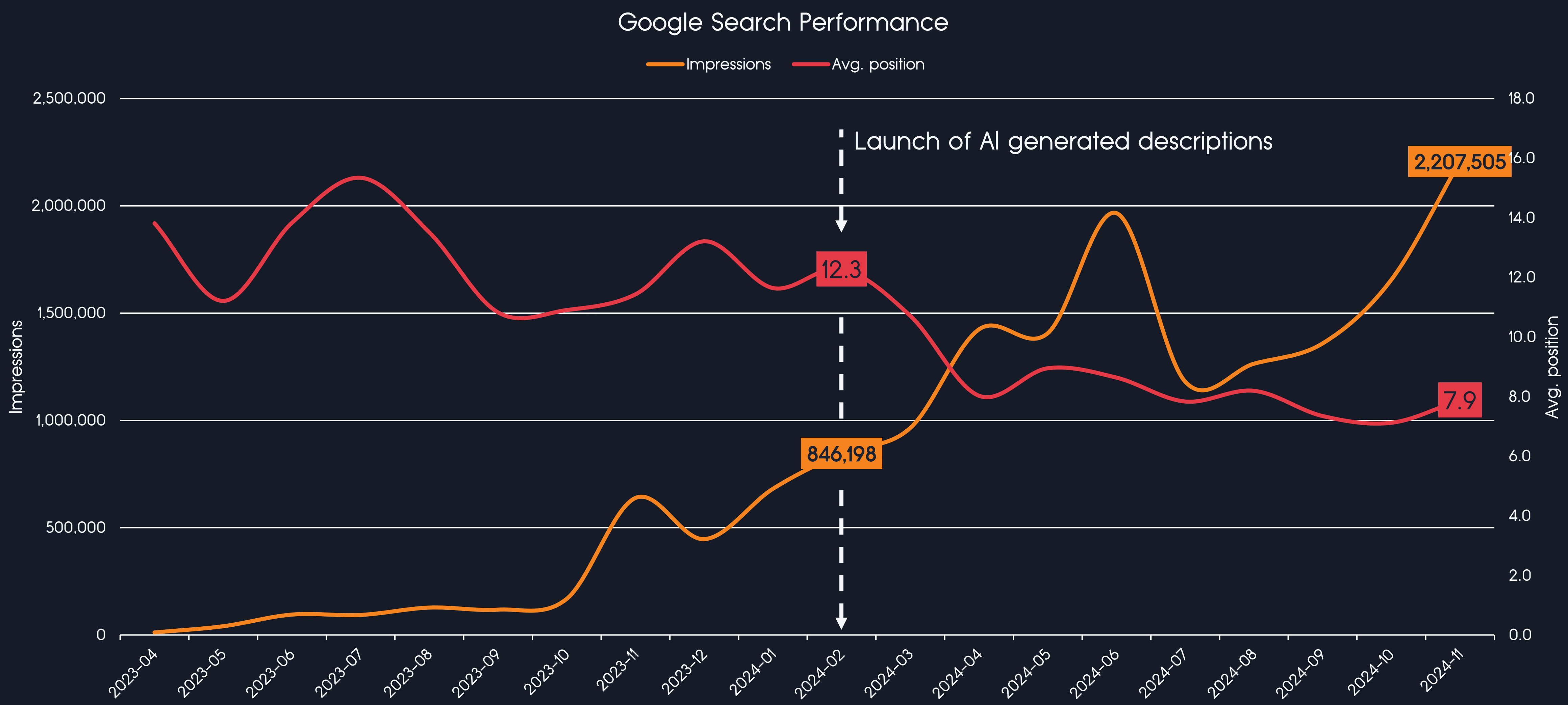
Per pastaruosius trejus metus UAB "Thermo Fisher Scientific Baltics" labai išaugo: turtas padidėjo nuo 2 262 672 007 eurų 2021 m. iki 12 511 353 990 eurų 2023 m., o tai yra įspūdingas 74,8 proc. metinis augimas. Nuosavo kapitalo bazė taip pat gerokai padidėjo - nuo 1 858 137 993 eurų 2021 m. iki 11 856 325,535 eurų 2023 m., o tai lėmė didelis pelningumas ir apdairus kapitalo valdymas.

Kalbant apie pajamas, Įmonė Thermo Fisher Scientific Baltics, UAB pranešė, kad 2021 m. gavo 1 941 513 774 EUR pardavimo pajamų, kurios 2022 m. per metus išaugo 53,7 %, o 2023 m. sumažėjo 44,4 % ir pasiekė 8 209 796 242 EUR. Nepaisant šio nuosmukio, UAB "Thermo Fisher Scientific Baltics" pelnas prieš mokesčius išliko didelis ir 2023 m. siekė 39 289 235 EUR, o 2022 m. - 649 290 EUR.

Įmonės Thermo Fisher Scientific Baltics grynojo pelno marža nuolat didėjo - nuo 31,8 % 2021 m. iki 37,5 % 2023 m. Tai rodo Bendrovės gebėjimą išlaikyti sąnaudų drausmę, kartu skatinant aukščiausios klasės augimą. Nuosavo kapitalo grąža (ROE), dar vienas svarbus rodiklis, 2021 m. svyravo nuo 33,2 % iki 69,0 % 2022 m., o tai rodo skirtingą sverto lygį ir investicijų grąžą.

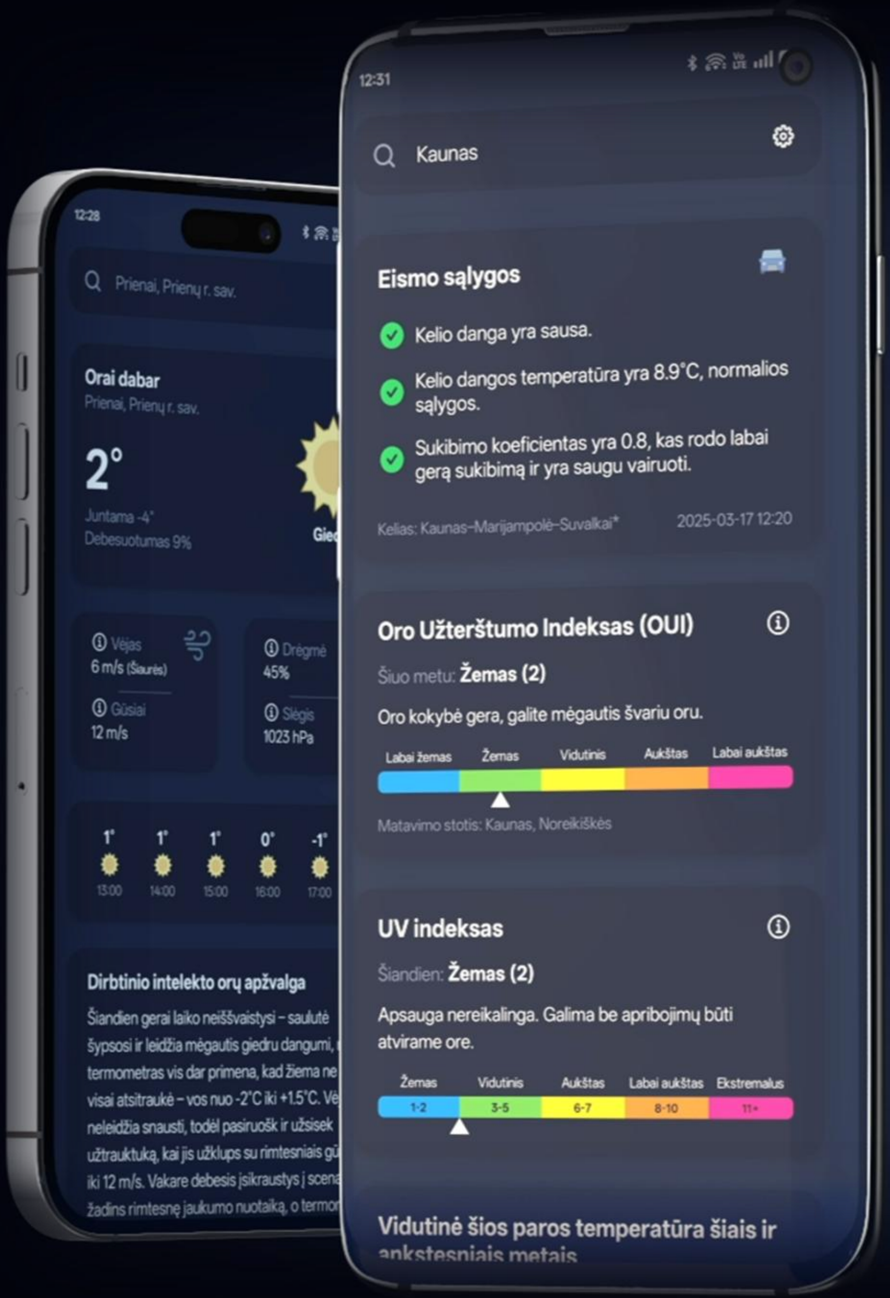
Apskritai, Įmonės Thermo Fisher Scientific Baltics finansiniai rezultatai rodo jos atsparumą ir gebėjimą prisitaikyti prie kintančių rinkos sąlygų. Turėdama tvirtą balansą ir didėjantį pelningumą, UAB "Thermo Fisher Scientific Baltics" yra gerai pasirengusi tolesnei sėkmei ateityje.

Did it work?


Google Search Performance
Impressions
Avg. position
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
18.0
16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0
Launch of AI generated descriptions
2,207,505
12.3
846,198
7.9
2023-04
2023-05
2023-06
2023-07
2023-08
2023-09
2023-10
2023-11
2023-12
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11

Main messages

- **Scoris** is probably the best Lithuanian business information website
- **Aphrodite engine** is probably the best local LLM inference engine for mass generation
- **Opus MT Seq2Seq** translation models are as good as Deepl and Google Translate (especially Scoris Fine-Tuned versions, available on HF)
  - *ONNX model format performs significantly faster than PyTorch model format*



Oriux - Orai Lietuvoje

Geriausia Lietuviška Orų Programėlė

Išbandykite jau dabar!

Download on the App Store

GET IT ON Google Play

www.oriux.lt

Thank you!

Find slides and files: ...