

Parallelizing the translation process

Parallelizing

Now that we have a function to translate a markdown file, we can parallelize the translation process.

First, we need to create a function to translate the front matter of a markdown file. This function will take the filename of the markdown file and the target language as input and return the translated front matter. We will use the `translate_markdown_file` function to translate the front matter of the markdown file.

Next, we will use the `concurrent.futures` module to parallelize the translation process. We will create a `ThreadPoolExecutor` with a maximum of 10 workers. We will then submit the `translate_markdown_file` function to the executor for each file in the `changed_files` list. Finally, we will wait for all the futures to complete and return the translated front matter.

Parallelizing

```
with concurrent.futures.ThreadPoolExecutor(max_workers=MAX_THREADS) as executor:
    futures = []
    for filename in changed_files:
        input_file = filename

        for lang in languages:

            print(f"Submitting translation job for {filename} to {lang}...")
            future = executor.submit(translate_markdown_file, input_file, os.path.join(f"_posts/{lang}", filename))
            futures.append(future)

    for future in concurrent.futures.as_completed(futures):
        try:
            future.result()
        except Exception as e:
            print(f"A thread failed: {e}")
```

Parallelizing

```
def translate_front_matter(front_matter, target_language, input_file):
    print(f" Translating front matter for: {input_file}")
```

```

if not front_matter:
    print(f" No front matter found for: {input_file}")
    return ""

try:
    front_matter_dict = {}
    if front_matter:
        front_matter_dict = yaml.safe_load(front_matter)
        print(f" Front matter after safe_load: {front_matter_dict}")
    if 'title' in front_matter_dict:
        print(f" Translating title: {front_matter_dict['title']}")
        if not (input_file == 'original/2025-01-11-resume-en.md' and target_language in ['zh', 'fr']):
            if isinstance(front_matter_dict['title'], str):
                translated_title = translate_text(front_matter_dict['title'], target_language)
                if translated_title:
                    translated_title = translated_title.strip()
                    if len(translated_title) > 300:
                        translated_title = translated_title.split('\n')[0]
                    front_matter_dict['title'] = translated_title
                    print(f" Translated title to: {translated_title}")
                else:
                    print(f" Title translation failed for: {input_file}")
            else:
                print(f" Title is not a string, skipping translation for: {input_file}")
        else:
            print(f" Skipping title translation for {input_file} to {target_language}")
    # Always set lang to target_language

    # Determine if the file is a translation
    original_lang = 'en' # Default to english
    if 'lang' in front_matter_dict:
        original_lang = front_matter_dict['lang']

    if target_language != original_lang:
        front_matter_dict['lang'] = target_language
        front_matter_dict['translated'] = True
        print(f" Marked as translated to {target_language} for: {input_file}")
    else:
        front_matter_dict['translated'] = False
        print(f" Not marked as translated for: {input_file}")

```

```

    result = "---\n" + yaml.dump(front_matter_dict, allow_unicode=True) + "---"
    print(f" Front matter translation complete for: {input_file}")
    return result
except yaml.YAMLError as e:
    print(f" Error parsing front matter: {e}")
    return front_matter

```

□□□□

```

def translate_front_matter(front_matter, target_language, input_file):
    print(f" Translating front matter for: {input_file}")
    if not front_matter:
        print(f" No front matter found for: {input_file}")
        return ""
    try:
        front_matter_dict = {}
        if front_matter:
            front_matter_dict = yaml.safe_load(front_matter)
            print(f" Front matter after safe_load: {front_matter_dict}")

        front_matter_dict_copy = front_matter_dict.copy()

        if 'title' in front_matter_dict_copy:
            print(f" Translating title: {front_matter_dict_copy['title']}")
            if not (input_file == 'original/2025-01-11-resume-en.md' and target_language in ['zh', 'fr']):
                if isinstance(front_matter_dict_copy['title'], str):
                    translated_title = translate_text(front_matter_dict_copy['title'], target_language)
                    if translated_title:
                        translated_title = translated_title.strip()
                        if len(translated_title) > 300:
                            translated_title = translated_title.split('\n')[0]
                        front_matter_dict_copy['title'] = translated_title
                        print(f" Translated title to: {translated_title}")
                    else:
                        print(f" Title translation failed for: {input_file}")
                else:
                    print(f" Title is not a string, skipping translation for: {input_file}")
            else:
                print(f" Skipping title translation for {input_file} to {target_language}")
    
```

```

# Always set lang to target_language

front_matter_dict_copy['lang'] = target_language
front_matter_dict_copy['translated'] = True

result = "---\n" + yaml.dump(front_matter_dict_copy, allow_unicode=True) + "---"
print(f" Front matter translation complete for: {input_file}")
return result
except yaml.YAMLError as e:
    print(f" Error parsing front matter: {e}")
    return front_matter

```