



adaptTo()

EUROPE'S LEADING AEM DEVELOPER CONFERENCE
28th – 30th SEPTEMBER 2026

Editorial AI in AEM

Giving Editors Control Over LLM-Generated Image Descriptions

José María Franco · Cognizant Netcentric · Vigo, Spain

adaptTo() 2026 · Berlin

Who's talking

José María Franco · Adobe Experience Cloud Technology Consultant, Cognizant Netcentric



- 25 years in software
- Technical lead for large consumers goods customer on the Adobe ecosystem: AEM Sites, Assets, Edge Delivery, Commerce
- Before: AEM as a Cloud Service across 60 markets, regulated healthcare
- Applied AI: LLM-assisted engineering inside enterprise governance constraints

The problem

A market needs to describe 2,000+ assets and asks us for a solution.

 SEO	Improves SEO by helping search engines understand image content and context
 Accessibility	WCAG 2.2 SC 1.1.1, and since the European Accessibility Act (June 2025), missing alt text can also be a legal issue with potential liability.

Doing this manually was just too slow and costly.

2,000 assets × 5 min = 167 hours ≈ 4 weeks of one person, full-time

The solution

First proposal on the table: generate descriptions from rules: filename, path, existing tags...

```
velo-pod-black-2023-FINAL-v3.jpg → "Velo pod black"  
IMG_4471.jpg → ?
```

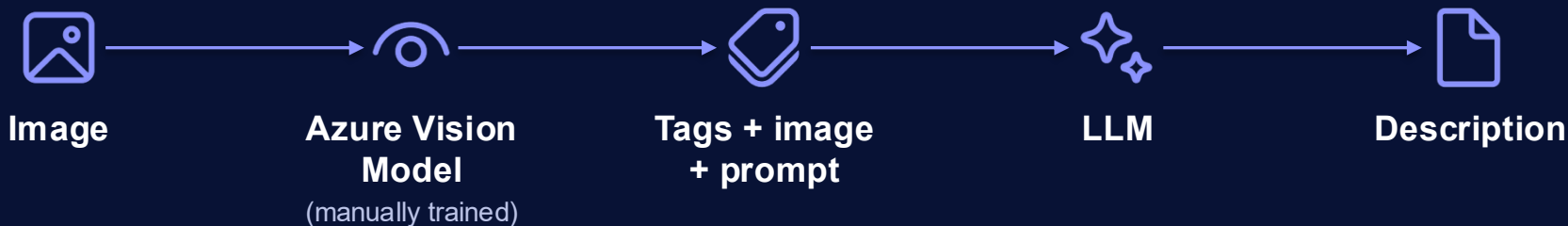
Rules can only rearrange metadata you already have.

If the description were derivable from the filename, you would not have the problem.

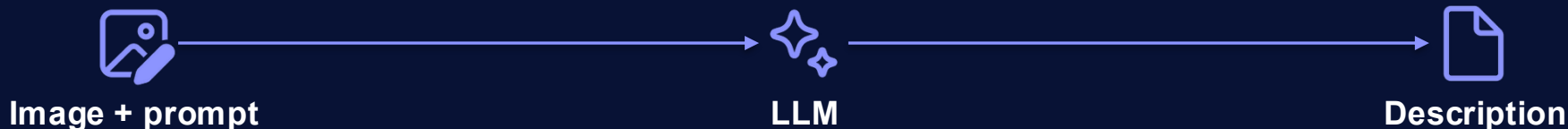
The image itself is the only reliable source. So: read the image.

The approach

What we designed (1st round):



What we shipped (final solution):



The tagging step made no measurable difference. The model did not need the scaffolding.

Advise: start simple. Avoid overengineering. Elaborate only if needed.

The challenge, part 1

The model works. The lever is the prompt.

```
One static prompt → "A black electronic device on a white background"
```

Accurate, but quite useless. We need more context, where the image lives on the site:

The image is used on...	What the description had to carry
Product detail page	Brand, model, category; the thing being sold
Campaign landing page	The offer, the call to action, what to do next
Support article	The feature or control being pointed at

Just a single prompt for everything was not enough

The challenge, part 2

What we actually needed

1. Prompt per purpose

product image · campaign asset · support screenshot

2. Prompt per brand, market and language

40+ markets, not just one English template translated

3. Marketers administering their own prompts

in a UI they already know

4. With real governance

permissions, versioning, review, rollback

Requirement 3 is the one that reframes the problem.

This stopped being just an AI integration and became a content-management challenge.

Architecture: where does the prompt live?

	Hardcoded in Java	OSGi config	DAM content
Who can change it	Developer	Developer / DevOps	Marketer
Requires deployment	Yes	No (config-as-code does)	No
Versioned	Git	Git	JCR versions
Reviewable before effect	PR	PR	Workflow
Per-language	if / else	One property each	One file each
Per-market ownership	No	No	Permissions
Can a marketer read it?	No	Sort of	Yes, it's a text file

Architecture: prompts as DAM assets

```
/content/dam/<project>/config/ai-prompts/  
├─ product-image-en.txt      ── campaign-en.txt  
├─ product-image-fr.txt     ── campaign-fr.txt  
├─ product-image-de.txt     ── support-screenshot-en.txt
```

`{promptsBasePath}/{promptType}-{language}.txt` - plain UTF-8. No templating. No DSL.

The prompt is content, so it inherits everything content already has:

versions & diffs · review workflow · permissions per market · MSM rollout

Benefits: No deployment. No developer involved. No ticket.

Architecture: platform settings

BAT - Azure OpenAI Alt Text Configuration

Configuration for Azure OpenAI alt text generation integration

Azure OpenAI API Key	***** Azure OpenAI API key for authentication (apiKey)
Azure OpenAI Endpoint	https://aem-image-description-resource-services.ai.azure.com/openai/v1/chat/completions Azure OpenAI endpoint URL (e.g., https://h-poc-azure-openai.openai.azure.com/) (endpoint)
Deployment Name	gpt-5.2 Azure OpenAI deployment/model name (e.g., gpt-5.2) (deploymentName)
Max Tokens	8192 Maximum number of tokens for alt text generation (maxTokens)
Temperature	 Temperature for AI generation (0.0-1.0, lower is more focused). Leave blank to omit from the request (API default will be used). Example: 0.2 (temperature)
Language Path Segment Index	5 0-based index of the language code segment in DAM paths (e.g., for /content/dam/batcom/markets/gb/en/..., index=5 extracts 'en') (languagePathSegmentIndex)
Default Language	en Default language code when path parsing fails (e.g., 'en', 'fr', 'de') (defaultLanguage)
Prompts DAM Base Path	/content/dam/batcom/config/ai-prompts DAM folder containing AI prompt text files (e.g., /content/dam/batcom/config/ai-prompts) (promptsBasePath)
Default Prompt Type	alt-text Default prompt type identifier (e.g., 'alt-text', 'product-image', 'lifestyle-image') (defaultPromptType)
Prompt Cache TTL (seconds)	5 Time-to-live for cached prompts in memory (promptCacheTtlSeconds)
Connection Timeout	5000 HTTP connection timeout in milliseconds (connectionTimeout)
Read Timeout	10000 HTTP read timeout in milliseconds (readTimeout)
Circuit Breaker Failure Threshold	5 Number of consecutive failures before opening circuit (circuitBreakerFailureThreshold)
Circuit Breaker Reset Timeout	60000 Time in milliseconds before attempting to close circuit after opening (circuitBreakerResetTimeout)

Configuration Information

Persistent Identity (PID)	com.bat.batcom.internal.services.alttext.AltTextGenerationServiceImpl
Configuration Binding	Unbound or new configuration

Cancel Reset Delete Unbind Save

Architecture: end-to-end workflow

Language from asset path • Prompt type from workflow argument



DAM payload

Asset or folder

Workflow step

Walk folders; skip
non-images and
existing descriptions

Generation service

Cached DAM
prompt
Original → base64
POST + circuit
breaker

Azure OpenAI

Vision + chat
completions

Asset metadata

dc:description

The workflow step

```
@Component(service = WorkflowProcess.class, property = { "process.label=Generate AI Alt Text" })
public class ImageAltTextWorkflowStep implements WorkflowProcess {

    public void execute(WorkItem workItem, WorkflowSession session, MetaDataMap metaDataMap) {
        // payload: single asset → process it
        //           folder      → findImageAssets() recursively
        // per asset: skip if not image/*
        //           skip if dc:description already set
        //           call altTextGenerationService
        // one session.save() at the end
    }
}
```

Skip rules are the feature, not the boilerplate.

The service: plain HTTP payload, no SDK

- Builds the call to the LLM endpoint, with the payload

```
{
  "model": "gpt-4",
  "max_completion_tokens": 125,
  "messages": [
    {
      "role": "system",
      "content": "You generate SEO-focused alt tags for
product images in an e-commerce catalog..."
    },
    {
      "role": "user",
      "content": [
        {
          "image_url": {
            "url":
"data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAABJgAAASYCAYAAABcV
HewAAAMh2lUWHR..."
          },
          "type": "image_url"
        }
      ]
    }
  ]
}
```

- Prompt loaded from the DAM via service user
→ cached in a ConcurrentHashMap with a TTL
- API key is `AttributeType.PASSWORD`, from the environment, never the repository

Apache HttpClient + jackson — both already on the AEM classpath.

Microsoft Foundry / aem-image-description / Deployments / gpt-5.2

New Foundry
All resources
aem-image-description-resource (swedencentral, SD)

Try the new Microsoft Foundry experience
Build and deploy your AI solutions faster, safer, and with the reliability you expect. [Learn more](#)

Start building →

Build and customize
Agents
Templates
Fine-tuning
Content Understanding
Observe and optimize
Tracing PREVIEW
Monitoring
Protect and govern
Evaluation PREVIEW
Guardrails + controls
Risks + alerts PREVIEW
Governance PREVIEW
Azure OpenAI
Stored completions (Deprecated)
Batch jobs
My assets
Data + indexes
Models + endpoints
More
Management center

gpt-5.2

Details
Metrics
Risks & Safety

Open in playground
Request quota
Edit
Delete

Endpoint

Target URI
<https://aem-image-description-resource.cognitiveservices.azure.com...>

Key

.....

Deployment info

Name gpt-5.2	Provisioning state Succeeded
Deployment type Global Standard	Created on 2026-08-28T15:19:34.746001Z
Created by josemaria.franco@netcentric.biz	Modified on Aug 28, 2026 5:19 PM
Modified by josemaria.franco@netcentric.biz	Version upgrade policy Once a new default version is available
Rate limit (Tokens per minute) 500,000	Rate limit (Requests per minute) 5,000
Model name	Model version

Language: Java
SDK: Azure OpenAI SDK
Authentication type: Key Authentication

Get Started

Below are example code snippets for a few use cases. For additional information about Azure OpenAI SDK, see full [documentation](#) and [samples](#).

1. Authentication using API Key

For OpenAI API Endpoints, deploy the Model to generate the endpoint URL and an API key to authenticate against the service. In this sample endpoint and key are strings holding the endpoint URL and the API Key.

The API endpoint URL and API key can be found on the Deployments + Endpoint page once the model is deployed.

To create a client with the OpenAI SDK using an API key, initialize the client by passing your API key to the SDK's configuration. This allows you to authenticate and interact with OpenAI's services seamlessly:

```

OpenAIClient client = new OpenAIClientBuilder()
    .credential(new AzureKeyCredential("<your-api-key>"))
    .endpoint("https://aem-image-description-resource.cognitiveservices.azure.com/");
    .buildClient();

```

2. Install dependencies

- To install, add this in your maven pom.xml:

Prompt anatomy

Minimal e-commerce oriented prompt example



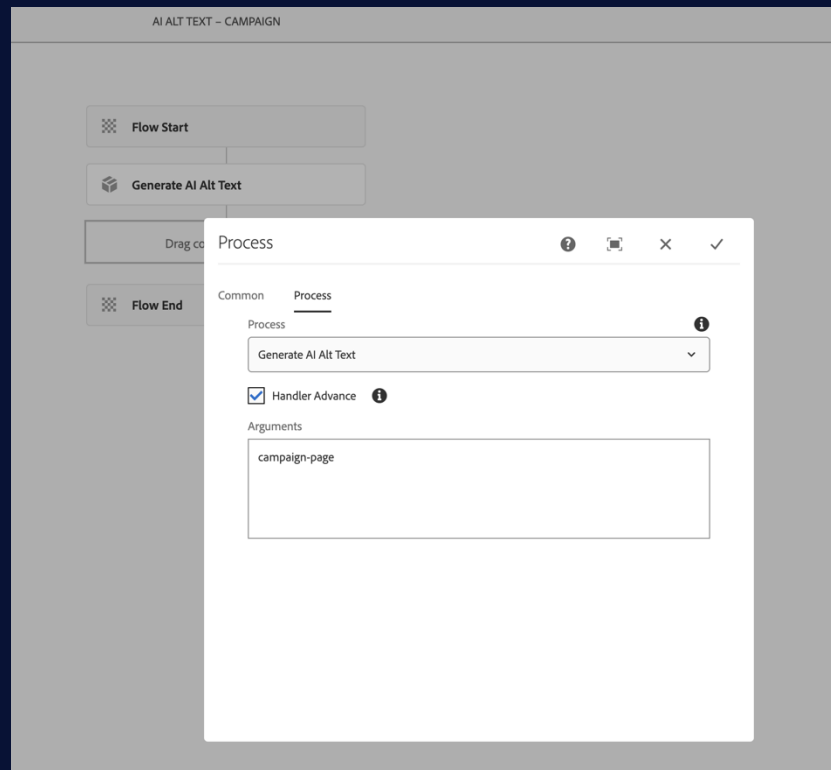
You generate concise, accurate alt text for consumer goods e-commerce images. Return ONLY the final alt text; never provide explanations, analysis, labels, or markdown. Output language: English. Generate the alt text directly in this language. For product images, prioritize brand, product/model, product type, variant, and relevant visible specifications. Include important specifications such as quantity, size, capacity, weight, or strength when clearly visible. For lifestyle or promotional images, describe the main meaningful subject and context, not every visual detail. Include meaningful text from the image when relevant, reproducing it verbatim; do not transcribe irrelevant text. Exclude legal, regulatory, and warning text unless it is essential to understanding the image. Write for accessibility first and SEO second: use natural language and avoid keyword stuffing. Never invent or infer information that cannot be reliably established from the image. If the image is purely decorative, return exactly "".

Thorough versions:

- Product pages: <https://github.com/jmffraiz/llm-alt-text/blob/main/prompts/ecommerce-prompt-en.md>
- Halloween lead-gen campaign: <https://github.com/jmffraiz/llm-alt-text/blob/main/prompts/halloween-loyalty-leadgen-prompt-en.md>

Workflow admin:

1. Creates a new workflow model
2. The argument is the asset that contains the prompt
 1. Just the name (without language code or extension)



A workflow model per prompt (purpose)

	Folder	Prompt	Workflow model
1	/.../en/products/	product-image-en.txt	Product Images
2	/.../fr/campaigns/	campaign-fr.txt	Campaign X Images

- DAM folders based on purpose or context
- Each folder is coupled with the right prompt for the purpose

What just happened



EN · product-image prompt
brand · model · category

Vuse PRO ONE Polar Mint device and pod kit, 20 mg/ml nicotine, black vape device with mint leaves, retail box packaging



FR · campaign prompt
description citing the campaign

Pods de cigarette électronique Vuse campagne X, vues rapprochées sur fond bleu glacé avec formes géométriques colorées

Two workflows --> Two different answers, directly in the right language, tailored to the right context

Production ready

Resilience

Circuit breaker	resilience4j — 50% failure rate over a sliding window, 60s open
Timeouts · pool	connect 5s · socket 10s · max 10 connections
Degradation	per-asset catch · one session.save() · failed assets simply stay empty

Failure mode = "nothing happened", never "half-written metadata".

Cost for the 2,000-asset batch

GPT-5 mini, per vision call

\$0.05

per asset

2,000 assets

\$100

done in under two hours

The manual alternative

167 h

4 weeks of one person

The next iteration: a human review gate

Today: Generate AI Alt Text → dc:description → published, indexed, read aloud



Generate AI Alt Text

AI-generated description

Participant Step

Marketer approves or edits

Publish

Published, indexed, read aloud

No new infrastructure. No new code. One step in a model that already exists.

The AI proposes, the editor publishes.

1

Don't reinvent the wheel — build on what AEM already gives you

The DAM manages the prompts and their whole lifecycle. Workflows are the execution UI. Permissions, versioning and review came with the address, not with our code.

2

Keep it simple — the models get better every day

Start with the naive version and measure. We built a two-step pipeline and the gain turned out to be negligible.

3

The line between config and content is thinner than you think

Our prompts are content. They live next to the images, where they belong — and that is what makes the author's life easy.

<https://github.com/jmffraiz/llm-alt-text>

José María Franco · [linkedin.com/in/jmfrancofraiz](https://www.linkedin.com/in/jmfrancofraiz) · josemaria.francofraiz@cognizant.com
<https://github.com/jmffraiz/llm-alt-text/blob/main/adaptto-2026-editorial-ai-in-aem.pdf>

