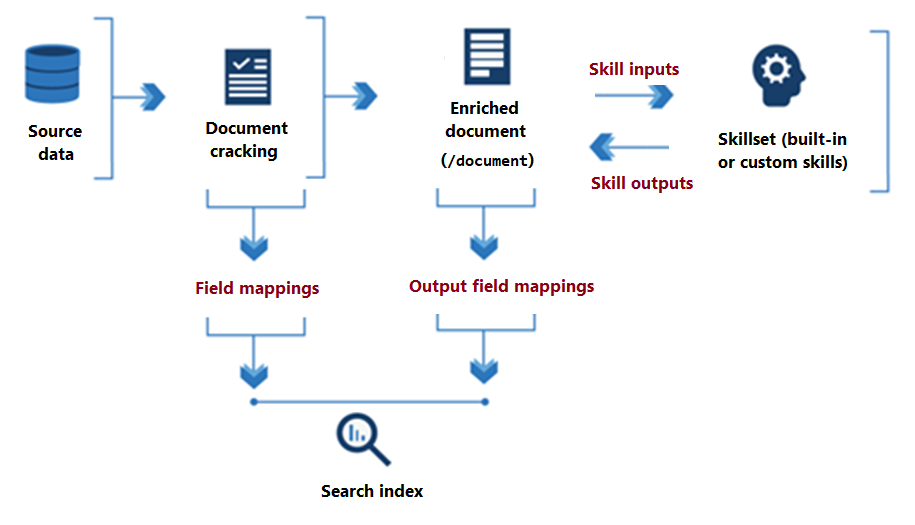
**Define a skillset**

A skillset is a reusable object in Azure AI Search that's attached to [an indexer](https://learn.microsoft.com/en-us/azure/search/search-indexer-overview). It contains one or more skills that call built-in AI or external custom processing over documents retrieved from an external data source.

The following diagram illustrates the basic data flow of skillset execution.



From the onset of skillset processing to its conclusion, skills read from and write to an [*enriched document*](https://learn.microsoft.com/en-us/azure/search/cognitive-search-working-with-skillsets?source=recommendations#enrichment-tree) that exists in memory. Initially, an enriched document is just the raw content extracted from a data source (articulated as the "/document" root node). With each skill execution, the enriched document gains structure and substance as each skill writes its output as nodes in the graph.

After skillset execution is done, the output of an enriched document finds its way into an index through user-defined *output field mappings*. Any raw content that you want transferred intact, from source to an index, is defined through *field mappings*.

To configure applied AI, specify settings in a skillset and indexer.

**Skillset definition**

A skillset is an array of one or more *skills* that perform an enrichment, such as translating text or optical character recognition (OCR) on an image file. Skills can be the [built-in skills](https://learn.microsoft.com/en-us/azure/search/cognitive-search-predefined-skills) from Microsoft, or [custom skills](https://learn.microsoft.com/en-us/azure/search/cognitive-search-create-custom-skill-example) for processing logic that you host externally. A skillset produces enriched documents that are either consumed during indexing or projected to a knowledge store.

Skills have a context, inputs, and outputs:



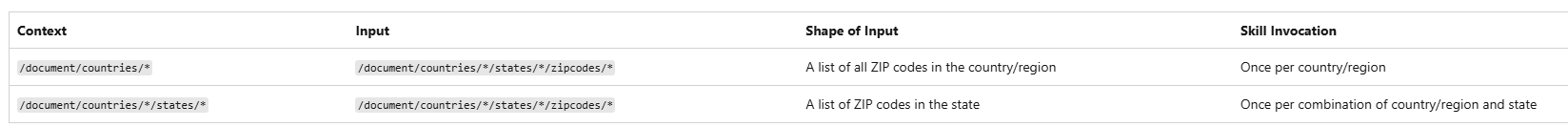
* [Context](https://learn.microsoft.com/en-us/azure/search/cognitive-search-working-with-skillsets?source=recommendations#skill-context) refers to the scope of the operation, which could be once per document or once for each item in a collection.
* Inputs originate from nodes in an enriched document, where a "source" and "name" identify a given node.
* Output is sent back to the enriched document as a new node. Values are the node "name" and node content. If a node name is duplicated, you can set a target name for disambiguation.

**Skill context**

Each skill has a context, which can be the entire document (/document) or a node lower in the tree (/document/countries/\*).

A context determines:

* The number of times the skill executes, over a single value (once per field, per document), or for a collection, where adding an /\* results in skill invocation for each instance in the collection.
* Output declaration, or where in the enrichment tree the skill outputs are added. Outputs are always added to the tree as children of the context node.
* Shape of the inputs. For multi-level collections, setting the context to the parent collection affects the shape of the input for the skill. For example if you have an enrichment tree with a list of countries/regions, each enriched with a list of states containing a list of ZIP codes, how you set the context determines how the input is interpreted.



**Skill dependencies**

Skills can execute independently and in parallel, or sequentially if you feed the output of one skill into another skill. The following example demonstrates two [built-in skills](https://learn.microsoft.com/en-us/azure/search/cognitive-search-predefined-skills) that execute in sequence:

* Skill #1 is a [Text Split skill](https://learn.microsoft.com/en-us/azure/search/cognitive-search-skill-textsplit) that accepts the contents of the "reviews\_text" source field as input, and splits that content into "pages" of 5,000 characters as output. Splitting large text into smaller chunks can produce better outcomes for skills like sentiment detection.
* Skill #2 is a [Sentiment Detection skill](https://learn.microsoft.com/en-us/azure/search/cognitive-search-skill-sentiment) accepts "pages" as input, and produces a new field called "Sentiment" as output that contains the results of sentiment analysis.

Notice how the output of the first skill ("pages") is used in sentiment analysis, where "/document/reviews\_text/pages/\*" is both the context and input. For more information about path formulation, see [How to reference enrichments](https://learn.microsoft.com/en-us/azure/search/cognitive-search-concept-annotations-syntax).

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

**Enrichment tree**

An enriched document is a temporary, tree-like data structure created during skillset execution that collects all of the changes introduced through skills. Collectively, enrichments are represented as a hierarchy of addressable nodes. Nodes also include any unenriched fields that are passed in verbatim from the external data source.

An enriched document exists for the duration of skillset execution, but can be [cached](https://learn.microsoft.com/en-us/azure/search/cognitive-search-incremental-indexing-conceptual) or sent to a [knowledge store](https://learn.microsoft.com/en-us/azure/search/knowledge-store-concept-intro).

Initially, an enriched document is simply the content extracted from a data source during [*document cracking*](https://learn.microsoft.com/en-us/azure/search/search-indexer-overview#document-cracking), where text and images are extracted from the source and made available for language or image analysis.

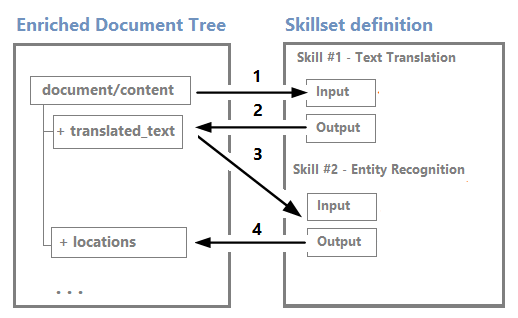
The initial content is metadata and the *root node* (document/content). The root node is usually a whole document or a normalized image that is extracted from a data source during document cracking. How it's articulated in an enrichment tree varies for each data source type. The following table shows the state of a document entering into the enrichment pipeline for several supported data sources:

A screenshot of a computer

Description automatically generated

As skills execute, output is added to the enrichment tree as new nodes. If skill execution is over the entire document, nodes are added at the first level under the root.

Nodes can be used as inputs for downstream skills. For example, skills that create content, such as translated strings, could become input for skills that recognize entities or extract key phrases.



Although you can visualize and work with an enrichment tree through the [Debug Sessions visual editor](https://learn.microsoft.com/en-us/azure/search/cognitive-search-debug-session), it's mostly an internal structure.

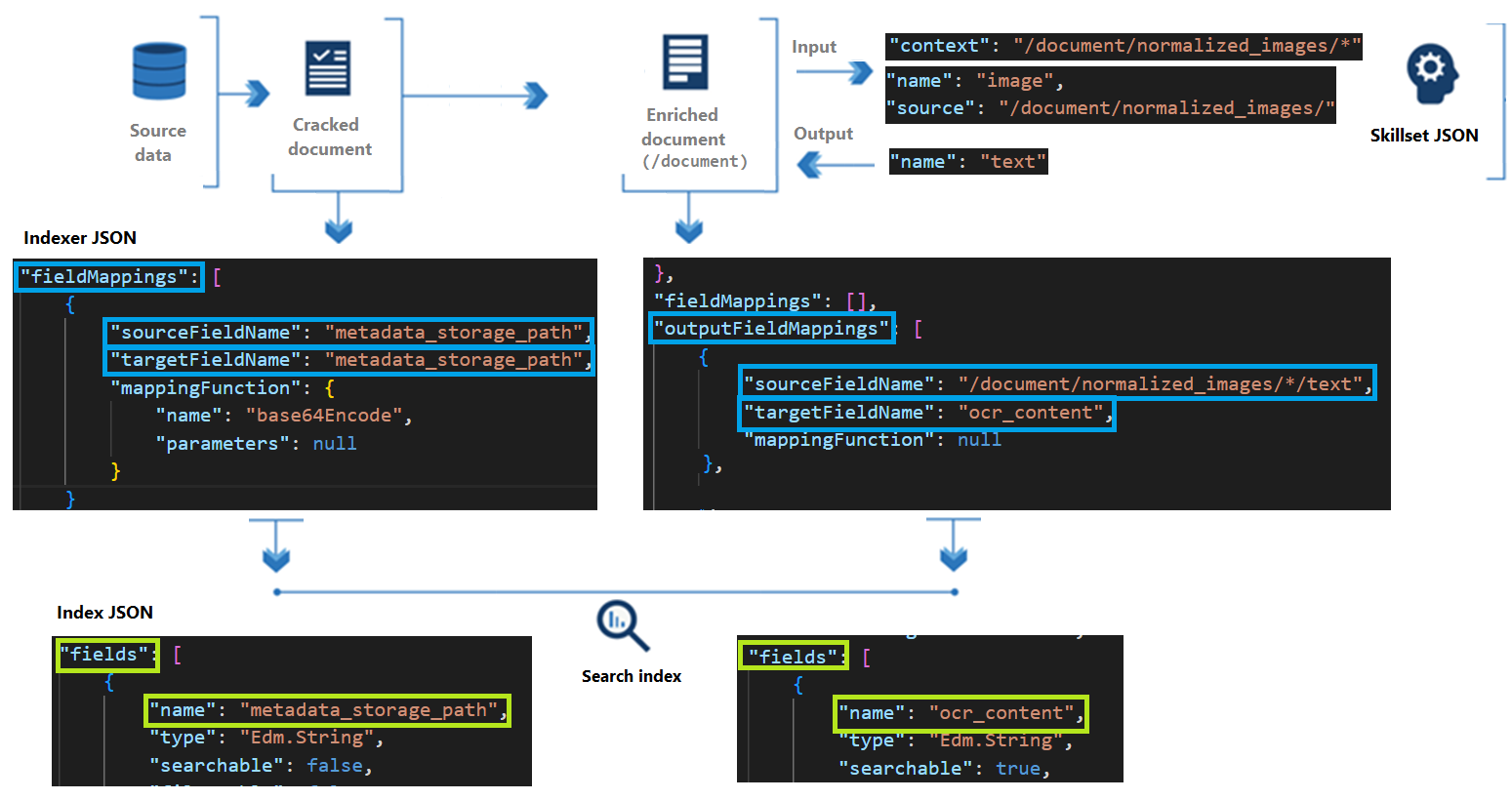
Enrichments are immutable: once created, nodes can't be edited. As your skillsets get more complex, so will your enrichment tree, but not all nodes in the enrichment tree need to make it to the index or the knowledge store.

You can selectively persist just a subset of the enrichment outputs so that you're only keeping what you intend to use. The [output field mappings](https://learn.microsoft.com/en-us/azure/search/cognitive-search-output-field-mapping) in your indexer definition determines what content actually gets ingested in the search index. Likewise, if you're creating a knowledge store, you can map outputs into [shapes](https://learn.microsoft.com/en-us/azure/search/knowledge-store-projection-shape) that are assigned to projections.

**Note :** The enrichment tree format enables the enrichment pipeline to attach metadata to even primitive data types. The metadata won't be a valid JSON object, but can be projected into a valid JSON format in projection definitions in a knowledge store.

**Indexer definition**

An indexer has properties and parameters used to configure indexer execution. Among those properties are mappings that set the data path to fields in a search index.

**[](https://learn.microsoft.com/en-us/azure/search/media/cognitive-search-working-with-skillsets/skillset-process-diagram-3.png#lightbox)**

There are two sets of mappings:

* ["fieldMappings"](https://learn.microsoft.com/en-us/azure/search/search-indexer-field-mappings) map a source field to a search field.
* ["outputFieldMappings"](https://learn.microsoft.com/en-us/azure/search/cognitive-search-output-field-mapping) map a node in an enriched document to a search field.

The "sourceFieldName" property specifies either a field in your data source or a node in an enrichment tree. The "targetFieldName" property specifies the search field in an index that receives the content.

**Enrichment example**

Using the [hotel reviews skillset](https://github.com/Azure-Samples/azure-search-sample-data/blob/main/hotelreviews/HotelReviews_skillset.json) as a reference point, this example explains how an [enrichment tree](https://learn.microsoft.com/en-us/azure/search/cognitive-search-working-with-skillsets#enrichment-tree) evolves through skill execution using conceptual diagrams.

This example also shows:

* How a skill's context and inputs work to determine how many times a skill executes
* What the shape of the input is based on the context

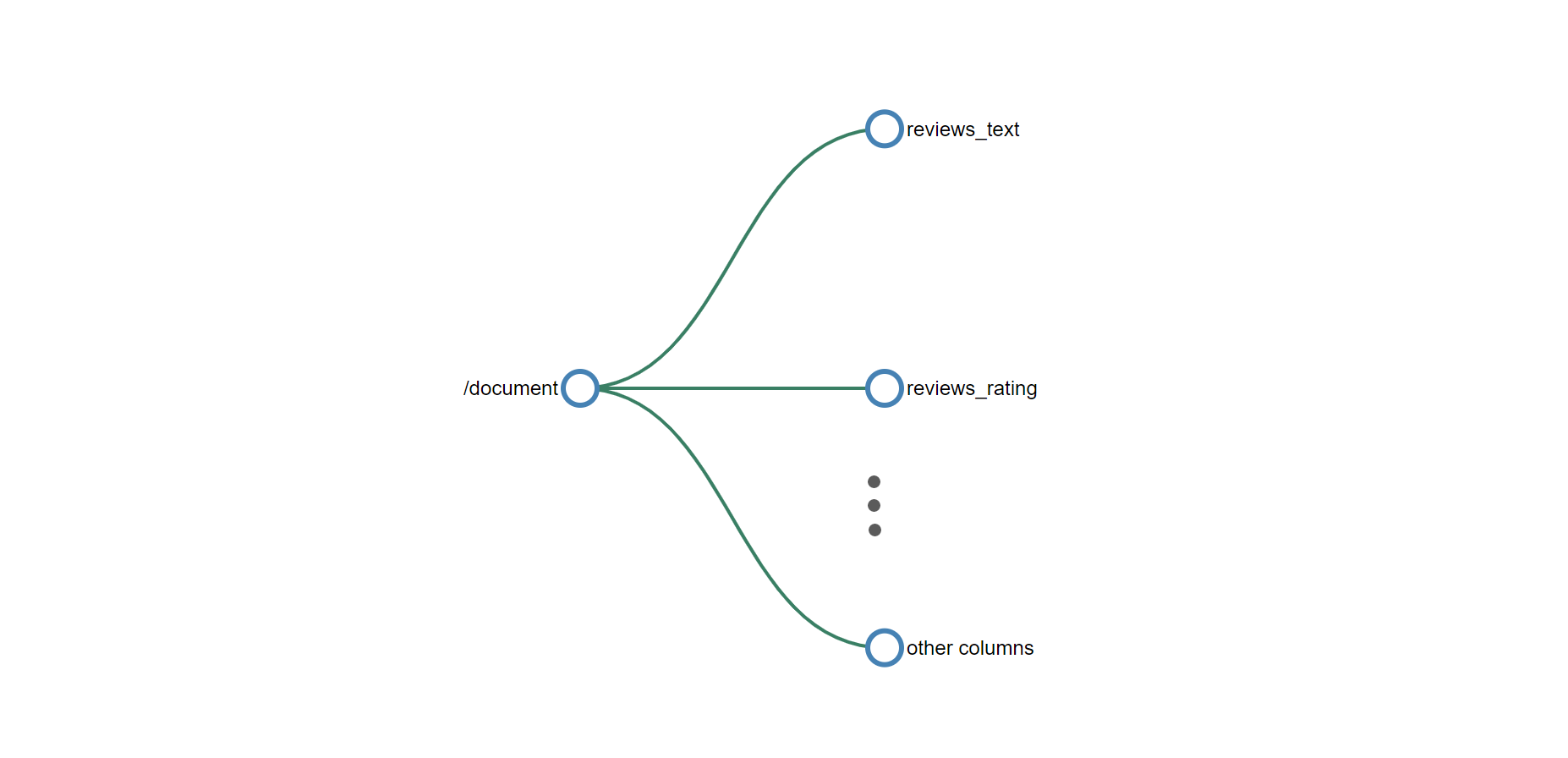
In this example, source fields from a CSV file include customer reviews about hotels ("reviews\_text") and ratings ("reviews\_rating"). The indexer adds metadata fields from Blob storage, and skills add translated text, sentiment scores, and key phrase detection.

In the hotel reviews example, a "document" within the enrichment process represents a single hotel review.

**Tip**

You can create a search index and knowledge store for this data in [**Azure portal**](https://learn.microsoft.com/en-us/azure/search/knowledge-store-create-portal) or [**REST APIs**](https://learn.microsoft.com/en-us/azure/search/knowledge-store-create-rest). You can also use [**Debug Sessions**](https://learn.microsoft.com/en-us/azure/search/cognitive-search-debug-session) for insights into skillset composition, dependencies, and effects on an enrichment tree. Images in this article are pulled from Debug Sessions.

Conceptually, the initial enrichment tree looks as follows:



The root node for all enrichments is "/document". When you're working with blob indexers, the "/document" node has child nodes of "/document/content" and "/document/normalized\_images". When the data is CSV, as in this example, the column names map to nodes beneath "/document".

**Skill #1: Split skill**

When source content consists of large chunks of text, it's helpful to break it into smaller components for greater accuracy of language, sentiment, and key phrase detection. There are two grains available: pages and sentences. A page consists of approximately 5,000 characters.

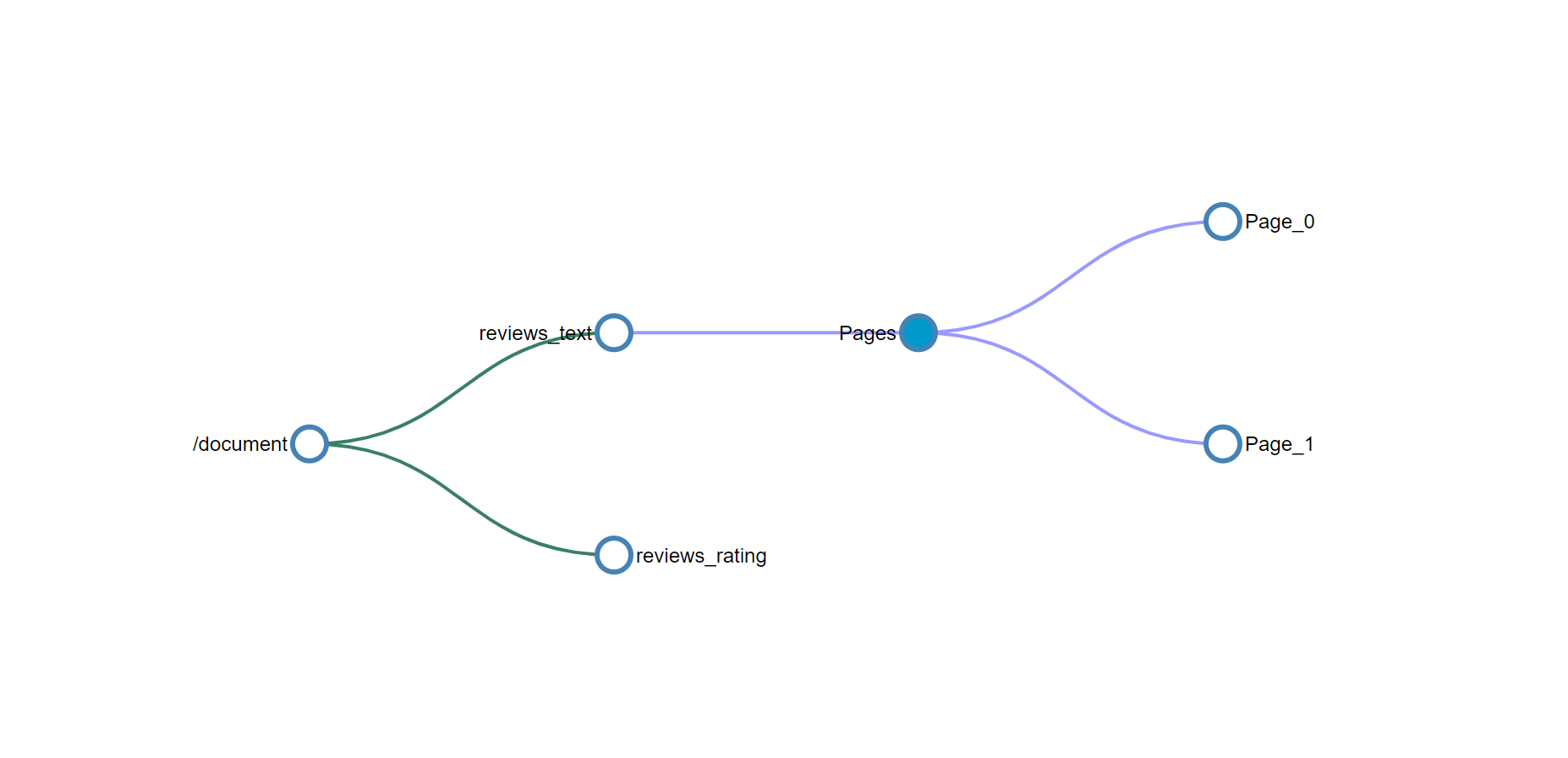
A text split skill is typically first in a skillset.

A screenshot of a computer program

Description automatically generated

With the skill context of "/document/reviews\_text", the split skill executes once for the reviews\_text. The skill output is a list where the reviews\_text is chunked into 5,000 character segments. The output from the split skill is named pages and it's added to the enrichment tree. The targetName feature allows you to rename a skill output before being added to the enrichment tree.

The enrichment tree now has a new node placed under the context of the skill. This node is available to any skill, projection, or output field mapping.

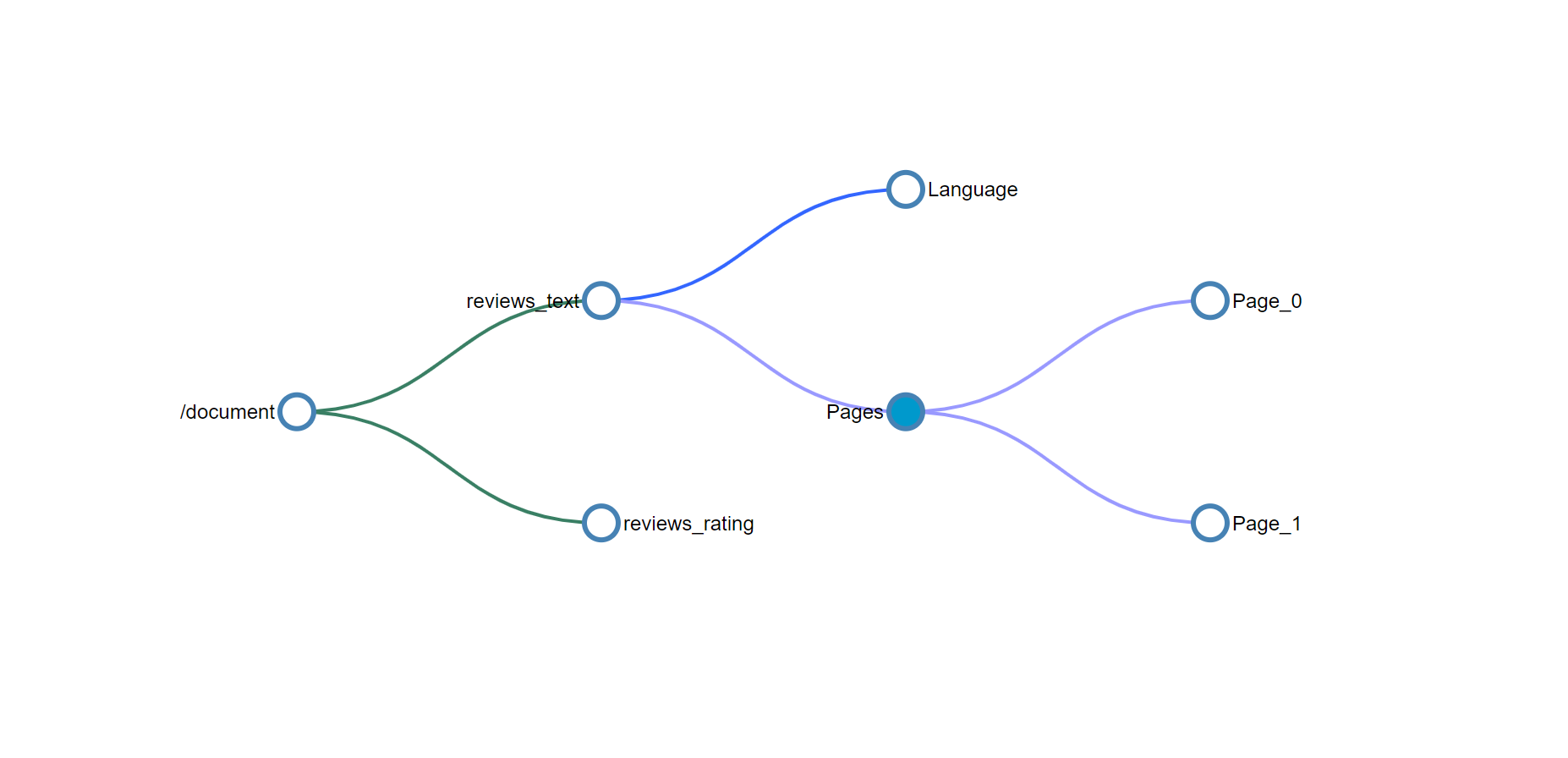


To access any of the enrichments added to a node by a skill, the full path for the enrichment is needed. For example, if you want to use the text from the pages node as an input to another skill, specify it as "/document/reviews\_text/pages/\*".

**Skill #2 Language detection**

Hotel review documents include customer feedback expressed in multiple languages. The language detection skill determines which language is used. The result will then be passed to key phrase extraction and sentiment detection (not shown), taking language into consideration when detecting sentiment and phrases.

While the language detection skill is the third (skill #3) skill defined in the skillset, it's the next skill to execute. It doesn't require any inputs so it executes in parallel with the previous skill. Like the split skill that preceded it, the language detection skill is also invoked once for each document. The enrichment tree now has a new node for language.

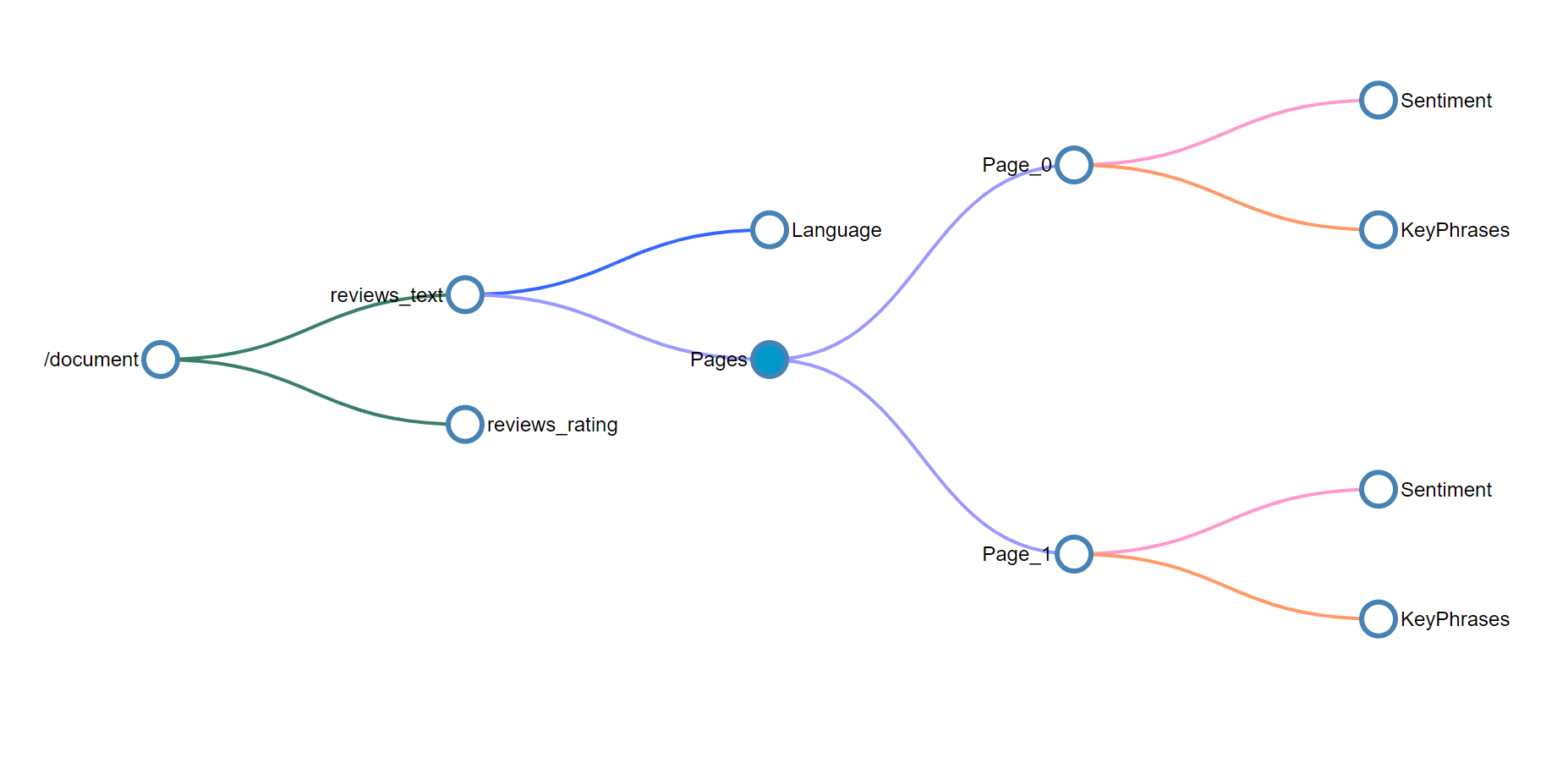


**Skills #3 and #4 (sentiment analysis and key phrase detection)**

Customer feedback reflects a range of positive and negative experiences. The sentiment analysis skill analyzes the feedback and assigns a score along a continuum of negative to positive numbers, or neutral if sentiment is undetermined. Parallel to sentiment analysis, key phrase detection identifies and extracts words and short phrases that appear consequential.

Given the context of /document/reviews\_text/pages/\*, both sentiment analysis and key phrase skills are invoked once for each of the items in the pages collection. The output from the skill will be a node under the associated page element.

You should now be able to look at the rest of the skills in the skillset and visualize how the tree of enrichments continue to grow with the execution of each skill. Some skills, such as the merge skill and the shaper skill, also create new nodes but only use data from existing nodes and don't create net new enrichments.



The colors of the connectors in the tree above indicate that the enrichments were created by different skills and the nodes need to be addressed individually and won't be part of the object returned when selecting the parent node.

**Skill #5 Shaper skill**

If output includes a [knowledge store](https://learn.microsoft.com/en-us/azure/search/knowledge-store-concept-intro), add a [Shaper skill](https://learn.microsoft.com/en-us/azure/search/cognitive-search-skill-shaper) as a last step. The Shaper skill creates data shapes out of nodes in an enrichment tree. For example, you might want to consolidate multiple nodes into a single shape. You can then project this shape as a table (nodes become the columns in a table), passing the shape by name to a table projection.

The Shaper skill is easy to work with because it focuses shaping under one skill. Alternatively, you can opt for in-line shaping within individual projections. The Shaper Skill doesn't add or detract from an enrichment tree, so it's not visualized. Instead, you can think of a Shaper skill as the means by which you rearticulate the enrichment tree you already have. Conceptually, this is similar to creating views out of tables in a database.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

**Skills for extra processing during indexing**

A *skill* provides an atomic operation that transforms content in some way. Often, it's an operation that recognizes or extracts text, but it can also be a utility skill that reshapes the enrichments that are already created. Typically, the output is text-based so that it can be used in [full text search](https://learn.microsoft.com/en-us/azure/search/search-lucene-query-architecture) or vectors used in [vector search](https://learn.microsoft.com/en-us/azure/search/vector-search-overview).

Skills are organized into categories:

* A ***built-in skill*** wraps API calls to an Azure AI resource, where the inputs, outputs, and processing steps are well understood. For skills that call an Azure AI resource, the connection is made over the internal network. For skills that call Azure OpenAI, you provide the connection information that the search service uses to connect to the resource. A small quantity of processing is non-billable, but at larger volumes, processing is billable. Built-in skills are based on pretrained models from Microsoft, which means you can't train the model using your own training data.
* A ***custom skill*** provides custom code that executes externally to the search service. It's accessed through a URI. Custom code is often made available through an Azure function app. To attach an open-source or third-party vectorization model, use a custom skill.
* A ***utility*** is internal to Azure AI Search, with no dependency on external resources or outbound connections. Most utilities are non-billable.

**Azure AI resource skills**

Skills that call the Azure AI are billed at the pay-as-you-go rate when you [attach an AI service resource](https://learn.microsoft.com/en-us/azure/search/cognitive-search-attach-cognitive-services).

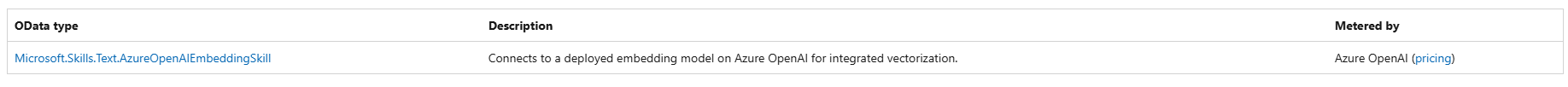
A close-up of a document

Description automatically generated



**Azure OpenAI skills**

Skills that call models deployed on Azure OpenAI are billed at the pay-as-you-go rate.



**Utility skills**

Skills that execute only on Azure AI Search, iterate mostly on nodes in the enrichment cache, and are mostly non-billable.

A close up of a computer screen

Description automatically generated

**Custom skills**

[Custom skills](https://learn.microsoft.com/en-us/azure/search/cognitive-search-custom-skill-web-api) wrap external code that you design, develop, and deploy to the web. You can then call the module from within a skillset as a custom skill.

