Moving away from hardcoded percentages (like 12% or 15%) to a **data-driven, dynamic approach** is exactly how enterprise-grade PDF parsers work. Because every document has different layouts—some have massive margins, others are packed edge-to-edge—you need algorithms that "learn" the layout of the specific page or document.

To do this using `page.get_text('dict')`, we define Spatial Zoning not by fixed boundaries, but by calculating the **"Document Body Core"** and identifying **Spatial Outliers** based on gaps, statistical distributions, and text characteristics.

Here is the advanced methodology for dynamic spatial zoning.

Phase 1: Establish the "Dominant Characteristics" (The Baseline)

Before you can identify an outlier (header/footer/side text), you must mathematically define what the "normal" body text is for that specific PDF.

1. **Extract Dominant Font Size:**
   Iterate through all spans on the page (or across the document). Create a frequency distribution of `span`, weighted by the number of characters in the span.
   - *Result:* The highest-weighted size is your `body_font_size` (e.g., 10.99 pt).
2. **Extract Standard Line Spacing:**
   Sort all lines by their `y0` coordinate. Calculate the vertical distance between consecutive lines (`next_line.y0 - current_line.y1`).
   - *Result:* The median of these positive distances is your `standard_line_gap`.
3. **Extract Primary Text Alignment (The Body Column):**
   For all lines that match the `body_font_size`, collect their `x0` (left edge) and `x1` (right edge) values.
   - Calculate the **Mode or 5th Percentile** of `x0`. This represents the true, dynamic left margin of the body text.
   - Calculate the **Mode or 95th Percentile** of `x1`. This is the true right margin.

Phase 2: Dynamic Horizontal Zoning (Left & Right Side Text)

Using `x0` and `x1` outliers is the most reliable way to find sidebars, vertical text, or marginalia.

The Algorithm:
Once you have your dynamic body column `(body_left_x0, body_right_x1)` from Phase 1, you evaluate every line on the page:

- **Left Side Text:** If a line's right edge (`line.x1`) is significantly strictly less than `body_left_x0` , it is a left-margin outlier.
- **Right Side Text:** If a line's left edge (`line.x0`) is significantly strictly greater than `body_right_x1` , it is a right-margin outlier.

Combining with Characteristics:
If an element falls outside the body column AND its `span` (direction) is rotated (e.g., `(0.0, -1.0)` for vertical text) OR its `span` != `body_font_size` , you have a 100% confidence match that this is noise/side-text to be removed.

Phase 3: Dynamic Vertical Zoning (Headers & Footers via Gap Analysis)

Instead of looking at the top 12%, we look for the **"Structural Chasms"** (large vertical gaps) that designers use to separate headers/footers from the main text.

The Algorithm (1D Clustering / Gap Detection):

1. Filter out the horizontal outliers (side text) so they don't mess up your vertical calculations.
2. Sort the remaining lines top-to-bottom by `y0` .
3. Calculate the gap to the next line.
4. **Detecting the Header Boundary:**
   Scan from the top line downwards. Look for a vertical gap that is **significantly larger** than the `standard_line_gap` (e.g., `gap > standard_line_gap * 2.5`).
   - Everything *above* this chasm is tentatively zoned as the Header region.
5. **Detecting the Footer Boundary:**
   Scan from the bottom line upwards. Look for the first massive vertical gap.
   - Everything *below* this chasm is tentatively zoned as the Footer region.

Phase 4: Validating Outliers with Text Characteristics (The Tie-Breaker)

Sometimes, a massive gap exists in the middle of a page (e.g., between two sections), or a title page has weird margins. You validate your spatial outliers using the `dict` characteristics to ensure you aren't deleting valid content.

A mathematically identified Header/Footer/Side-text region is confirmed if the lines inside it exhibit **Style Deviation**:

- **Size Deviation:** The `span` is smaller (e.g., 8pt disclaimer) or vastly larger (e.g., 24pt header) than the `body_font_size` .

- **Color Deviation:** The `span` is different (e.g., grey text `0x808080` while the body is black `0x000000`).
- **Block Isolation:** In `get_text('dict')`, the line belongs to a `block` that only contains 1 or 2 lines. (Body text blocks usually contain many lines).

How to Code This Logic (Mental Model)

Here is a pseudo-Python structure of how this advanced zoning engine operates:

```
import numpy as np

def analyze_page_layout(page_dict):
    lines = extract_all_lines(page_dict)

    # --- 1. BUILD STATISTICAL BASELINE ---
    # Extract all span sizes weighted by text length
    sizes = for line in lines for span in line]
    body_font_size = get_weighted_mode(sizes)

    # Find Body Column Edges (using lines that match body font)
    body_lines =
    left_x0_edge = np.percentile( for l in body_lines], 5)  # 5th percent
    right_x1_edge = np.percentile( for l in body_lines], 95) # 95th perce

    # Calculate Standard Line Spacing
    sorted_y_lines = sorted(body_lines, key=lambda l: l) # sort by y0
    gaps = - sorted_y_lines
           for i in range(len(sorted_y_lines)-1)]
    standard_gap = np.median() # Ignore overlapping lines

    # --- 2. EXECUTE ZONING ---
    zoned_lines =[]

    for line in lines:
        x0, y0, x1, y1 = line

        # Check Horizontal Outliers (Side Text)
        if x1 < left_x0_edge - 10: # 10 point buffer
            line = 'left_margin'
            continue
        if x0 > right_x1_edge + 10:
            line = 'right_margin'
```

```
            continue

        # Check Vertical Outliers (Gap Analysis)
        # (Implementation requires knowing the bounding box of the whole
        # but simplistically, check distance from the topmost/bottommost

        # Check Characteristics Deviation
        is_deviant_style = get_dominant_size(line) != body_font_size

        if y0 < (top_of_body_cluster - (standard_gap * 2)) and is_deviant
            line = 'header'
        elif y0 > (bottom_of_body_cluster + (standard_gap * 2)) and is_de
            line = 'footer'
        else:
            line = 'body'

        zoned_lines.append(line)

    return zoned_lines
```

Why this Advanced Approach Wins:

1. **Format Agnostic:** It works on A4, Letter, legal, or custom slide decks. It doesn't care about the actual page dimensions.
2. **Multi-Column Safe:** By looking at gaps and percentiles, you can adapt this to find multiple columns (by finding peaks in an `x0` histogram).
3. **Prevents Data Loss:** By relying heavily on **Style Deviation** (checking if the outlier text size/color actually differs from the body), you ensure that if an author simply leaves a huge gap between two standard body paragraphs, you won't accidentally delete the second paragraph thinking it's a footer.