

Supplementary Material S5. Code-Heuristic Arbitration Rules for Dual-LLM Cross-Validation

When the primary model Gemini 3 Pro and the control model Claude Opus 4.5 assign S_i values that differ by one tier or more on the same semantic indicator, the following code-heuristic arbitration rules are triggered. The arbitrators operate on the declarative properties of the CSS and the structural features of the visual layout, producing a deterministic anchor tier independent of any LLM output. All threshold parameters are derived from the corresponding clause boundaries defined in WCAG 2.2 itself; no scenario-specific expert ratings were used during their calibration. The final tier is the Gemini or Claude output whose distance to the code-evidence anchor is smaller, with ties broken toward the lower (more conservative) tier—consistent with Precedence Rule 1 of Section S3.5.

Pseudocode conventions. The functions below follow Python 3 syntax. Helper identifiers (extract_ui_components, compute_contrast, parse_interactive_elements, kendall_tau, extract_toggles, extract_text_elements, has_text_label_nearby, extract_numeric_displays) are implemented in Supplementary Material S1 (arbitration.py) and operate on the cleaned CSS produced by figma_css_cleaner.py. re is the Python standard-library regular-expressions module.

S5.1. V3 Non-text Contrast Arbitration

```
def arbitrate_v3(css_content):
    """
    Anchor by counting UI components whose foreground/background
    contrast falls below the SC 1.4.11 threshold of 3:1.
    """
    ui_components = extract_ui_components(css_content)
    low_contrast_count = sum(
        1 for c in ui_components
        if compute_contrast(c.bg, c.parent_bg) < 3.0
    )
    ratio = low_contrast_count / max(len(ui_components), 1)
    if ratio == 0:    return 100
    elif ratio < 0.20: return 75
    elif ratio < 0.50: return 50
    else:            return 25
```

S5.2. O3 Focus Order Arbitration

```
def arbitrate_o3(css_content):
    """
    Anchor by alignment between DOM declaration order and visual
    reading order (top-to-bottom, left-to-right). Measured by
    Kendall's tau between the two rankings.
```

```

"""
elements = parse_interactive_elements(css_content)
visual_order = sorted(elements, key=lambda e: (e.top, e.left))
dom_order = elements # parser returns declaration order
tau = kendall_tau(visual_order, dom_order)
if tau > 0.90: return 100
elif tau > 0.70: return 75
elif tau > 0.40: return 50
else: return 25

```

S5.3. A1 Status Messages Arbitration

```

def arbitrate_a1(css_content):
    """
    Anchor by coverage of textual or programmatic state markers
    adjacent to toggles, and presence of numeric status displays.
    The 100 tier in Table 2 requires a third audio channel, whose
    existence is a runtime property not derivable from static CSS;
    the arbitrator therefore caps its output at 75.
    """
    toggles = extract_toggles(css_content)
    texts = extract_text_elements(css_content)
    toggles_with_text = sum(
        1 for t in toggles if has_text_label_nearby(t, texts)
    )
    text_coverage = toggles_with_text / max(len(toggles), 1)
    has_numeric_status = bool(extract_numeric_displays(css_content))
    if text_coverage >= 0.80 and has_numeric_status:
        return 75
    elif text_coverage >= 0.50 or has_numeric_status:
        return 50
    else:
        return 25

```

S5.4. A2 Keyboard Accessibility Arbitration

```

def arbitrate_a2(css_content):
    """
    Anchor by presence of non-pointer input channels declared in
    the static markup: voice entries (Siri/microphone affordances)
    and keyboard-navigation primitives (tabindex, :focus, aria-label).
    """
    has_voice_entry = bool(re.search(
        r'siri|microphone|voice|\\bmic\\b',
        css_content, re.I
    ))
    has_keyboard_nav = bool(re.search(

```

```

        r'tabindex|:focus|aria-label',
        css_content, re.I
    ))
    if has_voice_entry and has_keyboard_nav: return 100
    elif has_voice_entry or has_keyboard_nav: return 75
    else: return 25

```

S5.5. Arbitration Driver

```

def arbitrate(metric_id, css_content, gemini_si, claude_si):
    """
    Triggered only when the two LLMs disagree by one tier or more.
    Returns the LLM output whose tier is closer to the anchor tier
    computed by the per-indicator arbitrator. Ties are broken
    toward the lower (more conservative) tier.
    """
    if gemini_si == claude_si:
        return gemini_si
    anchor = {
        'V3': arbitrate_v3,
        'O3': arbitrate_o3,
        'A1': arbitrate_a1,
        'A2': arbitrate_a2,
    }[metric_id](css_content)
    dist_g = abs(gemini_si - anchor)
    dist_c = abs(claude_si - anchor)
    if dist_g < dist_c:
        return gemini_si
    elif dist_c < dist_g:
        return claude_si
    else:
        return min(gemini_si, claude_si)

```