

Supplementary Data and Code

Fuzzy-DATA: A Strength- and Certainty-Calibrated Fuzzy TOPSIS Framework for Decision Support in Differentiated Thyroid Cancer

Doğan ŞENGÜL

İstanbul Sabahattin Zaim University, Faculty of Engineering and Natural Sciences, Department of Software Engineering, 34303, İstanbul, Türkiye

Clinical disclaimer: The framework and its outputs must not be used as patient-specific decisions without local calibration and clinician validation.

This material is provided for academic use only. If the data, the linguistic mappings, the calculation logic or the code in this document is used, the article above should be cited.

Intellectual Property & Usage Policy

This script is the intellectual property of the author and is developed strictly for academic and research purposes.

Attribution

Any usage, modification, adaptation or distribution of this code must be properly referenced and attributed to the author.

Disclaimer of Liability

No responsibility or liability is assumed for any commercial application, real-life implementation or operational usage of this code. This software is provided "as is" without warranty of any kind, express or implied. The author shall not be held liable for any damages or consequences arising from the use of this script outside of its intended academic scope.

This supplement compiles the prespecified fuzzy encodings, the complete fuzzy input matrices for both showcases, the baseline fuzzy *TOPSIS* results, the model ablation and one-way crossover results, the conditional numerical robustness summaries, the executable eligibility-gate test logs, the environment specification with fixed random seeds and the complete numerical-analysis reproduction script. The Recommendation 15 scope-mapping, Dirichlet-concentration and input-provenance tables are in the manuscript as Tables 7, 8 and 10. It contains no patient records, no identifiable information and no clinical outcome data. The plotting utilities that generate the figures are omitted from this listing because of space constraints.

S1. Prespecified triangular fuzzy encodings

Recommendation strength: Strong (0.80, 0.95, 1.00); Conditional (0.45, 0.65, 0.85); Good Practice Statement (0.80, 0.95, 1.00), applied without certainty multiplication; No recommendation (0.00, 0.20, 0.40). Evidence certainty: High (0.85, 0.95, 1.00); Moderate (0.65, 0.80, 0.90); Low (0.40, 0.60, 0.75); Very low (0.20, 0.40, 0.60); Insufficient (0.00, 0.20, 0.40). Applicability: Exact (0.85, 0.95, 1.00); Partial (0.45, 0.65, 0.85); Poor action/dose match (0.15, 0.35, 0.55). Performance ratings: Very low (0.00, 0.00, 0.25); Low (0.00, 0.25, 0.50); Moderate (0.25, 0.50, 0.75); High (0.50, 0.75, 1.00); Very high (0.75, 1.00, 1.00).

S2. Complete fuzzy input matrices (primary input)

Showcase 1: management of cT1aN0M0 papillary thyroid carcinoma.

Alternative	C1 guideline	C2 control	C3 harm	C4 pref.	C5 feas.	C6 monit.	C7 res./equity
Active surveillance	(0.153, 0.370, 0.637)	(0.500, 0.750, 1.000)	(0.750, 1.000, 1.000)	(0.750, 1.000, 1.000)	(0.750, 1.000, 1.000)	(0.750, 1.000, 1.000)	(0.500, 0.750, 1.000)
Thermal ablation	(0.153, 0.370, 0.637)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.000, 0.250, 0.500)	(0.250, 0.500, 0.750)	(0.000, 0.250, 0.500)
Thyroid lobectomy	(0.234, 0.494, 0.765)	(0.750, 1.000, 1.000)	(0.250, 0.500, 0.750)	(0.000, 0.250, 0.500)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.250, 0.500, 0.750)

Showcase 2: postoperative radioactive iodine selection in high-risk disease.

Alternative	C1 guideline	C2 control	C3 harm	C4 pref.	C5 feas.	C6 monit.	C7 res./equity
No RAI (surveillance only)	(0.085, 0.228, 0.480)	(0.000, 0.250, 0.500)	(0.750, 1.000, 1.000)	(0.000, 0.250, 0.500)	(0.750, 1.000, 1.000)	(0.250, 0.500, 0.750)	(0.750, 1.000, 1.000)
Low-activity RAI (30-50 mCi)	(0.078, 0.266, 0.495)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)
Adjuvant RAI (100-150 mCi)	(0.442, 0.722, 0.900)	(0.750, 1.000, 1.000)	(0.250, 0.500, 0.750)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.500, 0.750, 1.000)	(0.250, 0.500, 0.750)

S3. Baseline fuzzy TOPSIS results

Values are rounded to four decimal places here:

Showcase	Rank	Alternative	D+	D–	CC
Showcase 1	1	Active surveillance	0.3139	0.7651	0.7091
Showcase 1	2	Thyroid lobectomy	0.4548	0.6324	0.5817
Showcase 1	3	Thermal ablation	0.4930	0.5998	0.5488

Showcase 2	1	Adjuvant RAI (100-150 mCi)	0.3410	0.7432	0.6855
Showcase 2	2	Low-activity RAI (30-50 mCi)	0.4271	0.6648	0.6089
Showcase 2	3	No RAI (surveillance only)	0.5577	0.5219	0.4834

S4. Model ablation and one-way crossover thresholds

Showcase	Model	Guideline weight	Alternative	Rank	CC
Showcase 1	Context only	0.0	Active surveillance	1	0.8111
Showcase 1	Context only	0.0	Thyroid lobectomy	2	0.6097
Showcase 1	Context only	0.0	Thermal ablation	3	0.5966
Showcase 1	Proposed baseline	0.25	Active surveillance	1	0.7091
Showcase 1	Proposed baseline	0.25	Thyroid lobectomy	2	0.5817
Showcase 1	Proposed baseline	0.25	Thermal ablation	3	0.5488
Showcase 1	Guideline only	1.0	Thyroid lobectomy	1	0.4980
Showcase 1	Guideline only	1.0	Active surveillance	2	0.4029
Showcase 1	Guideline only	1.0	Thermal ablation	3	0.4029
Showcase 2	Context only	0.0	Low-activity RAI (30-50 mCi)	1	0.7066
Showcase 2	Context only	0.0	Adjuvant RAI (100-150 mCi)	2	0.6933
Showcase 2	Context only	0.0	No RAI (surveillance only)	3	0.5460
Showcase 2	Proposed baseline	0.25	Adjuvant RAI (100-150 mCi)	1	0.6855
Showcase 2	Proposed baseline	0.25	Low-activity RAI (30-50 mCi)	2	0.6089
Showcase 2	Proposed baseline	0.25	No RAI (surveillance only)	3	0.4834
Showcase 2	Guideline only	1.0	Adjuvant RAI (100-150 mCi)	1	0.6618
Showcase 2	Guideline only	1.0	Low-activity RAI (30-50 mCi)	2	0.3068
Showcase 2	Guideline only	1.0	No RAI (surveillance only)	3	0.2919

Primary approximate rank-reversal thresholds on a 0.0001 grid are 0.6796 for active surveillance versus thyroid lobectomy in Showcase 1 and 0.0370 for adjuvant versus low-activity radioactive iodine in Showcase 2. The Showcase 1 value uses the scope-qualified partial Recommendation 15 proxy and is a model-specific sensitivity result rather than a guideline-derived threshold.

S5. Conditional numerical robustness summaries (30,000 iterations per showcase)

Showcase	Alternative	Cond. top-rank freq.	Mean CC	SD CC	P05 CC	P95 CC
Showcase 1	Active surveillance	0.9925	0.7424	0.0403	0.6745	0.8069
Showcase 1	Thyroid lobectomy	0.0069	0.5958	0.0443	0.5223	0.6679
Showcase 1	Thermal ablation	0.0006	0.5593	0.0455	0.4848	0.6354
Showcase 2	Adjuvant RAI (100-150 mCi)	0.9274	0.7177	0.0396	0.6515	0.7820
Showcase 2	Low-activity RAI (30-50 mCi)	0.0726	0.6325	0.0434	0.5611	0.7043
Showcase 2	No RAI (surveillance only)	0.0000	0.4785	0.0435	0.4058	0.5492

Conditional top-rank frequency is the proportion of numerical stress-test iterations in which an alternative ranked first under the specified triangular and Dirichlet sampling assumptions. It is not a probability of treatment success or clinical superiority.

Showcase	Metric	Value
Showcase 1	Mean top-versus-second margin	0.1357
Showcase 1	P05 top-versus-second margin	0.0449
Showcase 1	Median top-versus-second margin	0.1364
Showcase 1	P95 top-versus-second margin	0.2232
Showcase 2	Mean top-versus-second margin	0.0889
Showcase 2	P05 top-versus-second margin	0.0106
Showcase 2	Median top-versus-second margin	0.0857

Showcase 2	P95 top-versus-second margin	0.1811
------------	------------------------------	--------

S5a. Executable eligibility-gate tests

The code applies active-surveillance eligibility predicates before ranking, records the decision and reason and tests an eligible case, an ineligible case and both sides of the 3-mm growth boundary. The suite also exercises every individual predicate and two combined-failure scenarios; the predicates are guideline-informed author operationalisations of ATA Recommendation 14 as documented in the script and in the *Safety Gates and Dynamic Updating* subsection of the manuscript.

Test case	Expected eligible	Observed eligible	Pass
Eligible baseline	True	True	True
Ineligible: unreliable follow-up	False	False	True
Boundary below threshold: 2.9 mm growth	True	True	True
Boundary at threshold: 3.0 mm growth	False	False	True
Ineligible: regional or distant metastases	False	False	True
Ineligible: gross extrathyroidal extension	False	False	True
Ineligible: posterior progression threatening critical structures	False	False	True
Ineligible: aggressive features	False	False	True
Ineligible: informed preference for surgery	False	False	True
Combined failure: unreliable follow-up and 3.2 mm growth	False	False	True
Combined failure: metastases, gross extension and aggressive features	False	False	True

Post-exclusion ranking when “reliable follow-up is unavailable”:

Rank	Alternative	CC
1	Thyroid lobectomy	0.5817
2	Thermal ablation	0.5488

S6. Environment and Fixed Random Seeds

Python 3.11 or later with numpy ≥ 1.24 , pandas ≥ 2.0 and matplotlib ≥ 3.7 . Fixed random seeds are 202608061 for Showcase 1 and 202608062 for Showcase 2. The primary Dirichlet concentration is 150 (with sensitivity at 50, 100, 300).

S7. Script

```
"""Simulation for the Fuzzy-DATA manuscript.
```

```
This script implements a strength- and certainty-calibrated fuzzy TOPSIS
framework for two differentiated thyroid cancer decision showcases.
```

```
It uses no patient-level data. All profiles, ratings and weights are illustrative.
```

```
This material is provided for academic use only.
```

```
If the data, the linguistic mappings, the calculation logic or the code in this document is used, the article above should be cited.
```

```
Copyright (c) 2026 Şengül, D.
```

```
INTELLECTUAL PROPERTY & USAGE POLICY
```

```
This script is the intellectual property of the author and is developed
strictly for academic and research purposes.
```

```
ATTRIBUTION
```

```
Any usage, modification, adaptation or distribution of this code must be
properly referenced and attributed to the author.
```

DISCLAIMER OF LIABILITY

No responsibility or liability is assumed for any commercial application, real-life implementation or operational usage of this code. This software is provided "as is" without warranty of any kind, express or implied. The author shall not be held liable for any damages or consequences arising from the use of this script outside of its intended academic scope.

Python: 3.11+

Dependencies for the reproduced numerical analyses: numpy and pandas. Matplotlib was used separately for figure preparation; the plotting utilities are not included in this listing.

"""

```
from _future_ import annotations
```

```
import math
```

```
from pathlib import Path
```

```
from typing import Dict, Iterable, List, Mapping, Sequence, Tuple
```

```
import numpy as np
```

```
import pandas as pd
```

```
TFN = Tuple[float, float, float]
```

```
# -----
```

```
# Fuzzy scales
```

```
# -----
```

```
STRENGTH: Mapping[str, TFN] = {
```

```
    "strong": (0.80, 0.95, 1.00),
```

```
    "conditional": (0.45, 0.65, 0.85),
```

```
    # A Good Practice Statement is not GRADE-d; it is represented directly.
```

```

    "good_practice": (0.80, 0.95, 1.00),
    "no_recommendation": (0.00, 0.20, 0.40),
}

```

```

CERTAINTY: Mapping[str, TFN] = {
    "high": (0.85, 0.95, 1.00),
    "moderate": (0.65, 0.80, 0.90),
    "low": (0.40, 0.60, 0.75),
    "very_low": (0.20, 0.40, 0.60),
    "insufficient": (0.00, 0.20, 0.40),
}

```

```

LINGUISTIC: Mapping[str, TFN] = {
    "VL": (0.00, 0.00, 0.25),
    "L": (0.00, 0.25, 0.50),
    "M": (0.25, 0.50, 0.75),
    "H": (0.50, 0.75, 1.00),
    "VH": (0.75, 1.00, 1.00),
}

```

```

CRITERIA: Sequence[str] = (
    "Guideline concordance",
    "Expected disease control",
    "Harm and burden avoidance",
    "Patient preference alignment",
    "Feasibility and acceptability",
    "Monitoring reliability",
    "Resource and equity fit",
)

```

```

BASE_WEIGHTS = np.array([0.25, 0.20, 0.15, 0.15, 0.10, 0.10, 0.05], dtype=float)
CONTEXT_PROPORTIONS = BASE_WEIGHTS[1:] / BASE_WEIGHTS[1:].sum()

```

```

def tfn_multiply(a: TFN, b: TFN) -> TFN:
    """Componentwise product for nonnegative triangular fuzzy numbers."""
    return tuple(float(a[k] * b[k]) for k in range(3)) # type: ignore[return-value]

```

```

def tfn_complement(a: TFN) -> TFN:
    """Standard complement on [0,1]."""
    lower, mode, upper = a
    return 1.0 - upper, 1.0 - mode, 1.0 - lower

```

```

def centroid(a: TFN) -> float:
    return float(sum(a) / 3.0)

```

```

def guideline_concordance(
    strength_label: str,
    certainty_label: str | None,
    applicability: TFN,
    direction: str = "for",
) -> TFN:
    """Create fuzzy guideline concordance from recommendation metadata.

```

For graded recommendations, strength and certainty are multiplied.
 Good Practice Statements are represented directly because they are not GRADE-d.

```

Direction='against' applies the fuzzy complement before applicability.
Applicability captures population, intervention, dose and setting match.
"""

if strength_label == "good_practice":
    base = STRENGTH[strength_label]
else:
    if certainty_label is None:
        raise ValueError("certainty_label is required for graded recommendations")
    base = tfn_multiply(STRENGTH[strength_label], CERTAINTY[certainty_label])
if direction == "against":
    base = tfn_complement(base)
elif direction != "for":
    raise ValueError("direction must be 'for' or 'against'")
return tfn_multiply(base, applicability)


def vertex_distance(a: TFN, b: TFN) -> float:
    """Chen-style vertex distance between triangular fuzzy numbers."""
    return math.sqrt(sum((a[k] - b[k]) ** 2 for k in range(3)) / 3.0)


def fuzzy_topsis(
    matrix: Mapping[str, Sequence[TFN]], weights: np.ndarray = BASE_WEIGHTS
) -> pd.DataFrame:
    """Run fuzzy TOPSIS for benefit-form criteria already normalized to [0,1]."""
    if not np.isclose(weights.sum(), 1.0):
        raise ValueError("weights must sum to 1")
    alternatives = list(matrix)
    weighted: Dict[str, List[TFN]] = {}
    for alternative in alternatives:

```

```

if len(matrix[alternative]) != len(weights):
    raise ValueError(f"{alternative} has an incorrect number of criteria")
weighted[alternative] = [
    tuple(float(weights[j] * value) for value in matrix[alternative][j])
    for j in range(len(weights))
]

positive_ideal = [(float(w), float(w), float(w)) for w in weights]
negative_ideal = [(0.0, 0.0, 0.0) for _ in weights]
rows = []
for alternative in alternatives:
    d_positive = sum(
        vertex_distance(weighted[alternative][j], positive_ideal[j])
        for j in range(len(weights))
    )
    d_negative = sum(
        vertex_distance(weighted[alternative][j], negative_ideal[j])
        for j in range(len(weights))
    )
    closeness = d_negative / (d_positive + d_negative)
    rows.append(
        {
            "Alternative": alternative,
            "D_positive": d_positive,
            "D_negative": d_negative,
            "Closeness_coefficient": closeness,
        }
    )
result = pd.DataFrame(rows).sort_values(
    "Closeness_coefficient", ascending=False, ignore_index=True
)
result.insert(0, "Rank", np.arange(1, len(result) + 1))

```

```

return result

def weights_for_guideline_share(lambda_guideline: float) -> np.ndarray:
    if not 0.0 <= lambda_guideline <= 1.0:
        raise ValueError("lambda_guideline must lie in [0,1]")
    return np.concatenate(
        ([lambda_guideline], (1.0 - lambda_guideline) * CONTEXT_PROPORTIONS)
    )

def sensitivity_analysis(
    matrix: Mapping[str, Sequence[TFN]], lambdas: Iterable[float]
) -> pd.DataFrame:
    rows = []
    for lambda_guideline in lambdas:
        result = fuzzy_topsis(matrix, weights_for_guideline_share(lambda_guideline))
        for _, row in result.iterrows():
            rows.append(
                {
                    "Guideline_weight": lambda_guideline,
                    "Alternative": row["Alternative"],
                    "Closeness_coefficient": row["Closeness_coefficient"],
                    "Rank": int(row["Rank"]),
                }
            )
    return pd.DataFrame(rows)

def crisp_topsis(scores: np.ndarray, weights: np.ndarray) -> np.ndarray:

```

```
"""Sampled counterpart of the manuscript's summed criterion-level metric.
```

```
Each sampled TFN becomes a crisp score in [0, 1]. The positive and negative distances are calculated criterion by criterion and then summed, matching equations 6-9 used by ``fuzzy_topsis`` rather than applying one global Euclidean norm across criteria.
```

```
"""
```

```
weighted_scores = scores * weights
d_positive = np.abs(weighted_scores - weights).sum(axis=1)
d_negative = np.abs(weighted_scores).sum(axis=1)
return d_negative / (d_positive + d_negative)
```

```
def monte_carlo(
    matrix: Mapping[str, Sequence[TFN]],
    iterations: int,
    concentration: float,
    seed: int,
    retain_draws: bool = True,
) -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    """Run a conditional numerical robustness simulation.
```

```
The triangular draws are a transparent stress-test device that perturbs each TFN within its stated bounds with greatest density near its modal value; they are not empirical probabilities and do not convert fuzzy membership into a probability model. Criterion weights are sampled from Dirichlet(concentration * BASE_WEIGHTS), preserving the baseline means and unit sum. Reported rank frequencies are conditional on these assumptions.
```

```
"""
```

```
rng = np.random.default_rng(seed)
```

```

alternatives = list(matrix)
n_alternatives = len(alternatives)
n_criteria = len(BASE_WEIGHTS)
wins = np.zeros(n_alternatives, dtype=int)
coefficients = np.zeros((iterations, n_alternatives), dtype=float)
margins = np.zeros(iterations, dtype=float)

for iteration in range(iterations):
    weights = rng.dirichlet(BASE_WEIGHTS * concentration)
    sampled_scores = np.zeros((n_alternatives, n_criteria), dtype=float)
    for i, alternative in enumerate(alternatives):
        for j, (lower, mode, upper) in enumerate(matrix[alternative]):
            sampled_scores[i, j] = rng.triangular(lower, mode, upper)
    cc = crisp_topsis(sampled_scores, weights)
    coefficients[iteration] = cc
    order = np.argsort(-cc)
    wins[order[0]] += 1
    margins[iteration] = cc[order[0]] - cc[order[1]]

summary = pd.DataFrame(
    {
        "Alternative": alternatives,
        "Conditional_top_rank_frequency": wins / iterations,
        "Mean_closeness": coefficients.mean(axis=0),
        "SD_closeness": coefficients.std(axis=0),
        "P05_closeness": np.quantile(coefficients, 0.05, axis=0),
        "P95_closeness": np.quantile(coefficients, 0.95, axis=0),
    }
).sort_values("Conditional_top_rank_frequency", ascending=False, ignore_index=True)

if retain_draws:

```

```

draw_rows = []
for iteration in range(iterations):
    for i, alternative in enumerate(alternatives):
        draw_rows.append(
            {
                "Iteration": iteration + 1,
                "Alternative": alternative,
                "Closeness_coefficient": coefficients[iteration, i],
            }
        )
    draws = pd.DataFrame(draw_rows)
else:
    draws = pd.DataFrame(
        columns=["Iteration", "Alternative", "Closeness_coefficient"]
    )
margin_summary = pd.DataFrame(
    {
        "Metric": [
            "Mean top-versus-second margin",
            "P05 top-versus-second margin",
            "Median top-versus-second margin",
            "P95 top-versus-second margin",
        ],
        "Value": [
            margins.mean(),
            np.quantile(margins, 0.05),
            np.quantile(margins, 0.50),
            np.quantile(margins, 0.95),
        ],
    }
)
return summary, draws, margin_summary

```

```

# -----
# Synthetic showcases
# -----
EXACT: TFN = (0.85, 0.95, 1.00)
PARTIAL: TFN = (0.45, 0.65, 0.85)
POOR_DOSE_MATCH: TFN = (0.15, 0.35, 0.55)

SHOWCASE_1: Dict[str, Sequence[TFN]] = {
    "Active surveillance": (
        guideline_concordance("conditional", "low", EXACT, "for"),
        LINGUISTIC["H"],
        LINGUISTIC["VH"],
        LINGUISTIC["VH"],
        LINGUISTIC["VH"],
        LINGUISTIC["VH"],
        LINGUISTIC["H"],
    ),
    "Thermal ablation": (
        guideline_concordance("conditional", "low", EXACT, "for"),
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["L"],
        LINGUISTIC["M"],
        LINGUISTIC["L"],
    ),
    "Thyroid lobectomy": (
        guideline_concordance("strong", "moderate", PARTIAL, "for"),
        LINGUISTIC["VH"],
    ),
}

```

```

    LINGUISTIC["M"],
    LINGUISTIC["L"],
    LINGUISTIC["H"],
    LINGUISTIC["H"],
    LINGUISTIC["M"],
),
}

SHOWCASE_2: Dict[str, Sequence[TFN]] = {
    "No RAI (surveillance only)": (
        guideline_concordance("strong", "moderate", EXACT, "against"),
        LINGUISTIC["L"],
        LINGUISTIC["VH"],
        LINGUISTIC["L"],
        LINGUISTIC["VH"],
        LINGUISTIC["M"],
        LINGUISTIC["VH"],
    ),
    "Low-activity RAI (30-50 mCi)": (
        guideline_concordance("strong", "moderate", POOR_DOSE_MATCH, "for"),
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["H"],
        LINGUISTIC["H"],
    ),
    "Adjuvant RAI (100-150 mCi)": (
        guideline_concordance("strong", "moderate", EXACT, "for"),
        LINGUISTIC["VH"],
        LINGUISTIC["M"],
        LINGUISTIC["H"],
    )
}

```

```

    LINGUISTIC["H"],
    LINGUISTIC["H"],
    LINGUISTIC["M"],
),
}

```

```

# Please note that recommendation 15 applies to surgical extent when resection is selected. The
# primary flat showcase therefore treats it as a scope-qualified partial proxy
# and explicitly stress-tests alternative mappings rather than presenting the
# resulting crossover as a guideline-derived threshold.

```

```

RECOMMENDATION_15_SCOPE_SCENARIOS: Mapping[str, TFN] = {
    "Scope-qualified partial proxy (primary)": guideline_concordance(
        "strong", "moderate", PARTIAL, "for"
    ),
    "Full direct transfer (stress test)": guideline_concordance(
        "strong", "moderate", EXACT, "for"
    ),
    "Parent-node harmonised (conditional/low)": guideline_concordance(
        "conditional", "low", EXACT, "for"
    ),
    "No direct Recommendation 15 credit at parent node": (0.0, 0.0, 0.0),
}

```

```

def showcase_1_with_lobectomy_guideline(guideline_rating: TFN) -> Dict[str, Sequence[TFN]]:
    revised = {alternative: tuple(ratings) for alternative, ratings in SHOWCASE_1.items()}
    lobectomy = list(revised["Thyroid lobectomy"])
    lobectomy[0] = guideline_rating
    revised["Thyroid lobectomy"] = tuple(lobectomy)

```

```
return revised
```

```
def recommendation_15_scope_analysis() -> pd.DataFrame:
    rows = []
    for scenario, guideline_rating in RECOMMENDATION_15_SCOPE_SCENARIOS.items():
        matrix = showcase_1_with_lobectomy_guideline(guideline_rating)
        result = fuzzy_topsis(matrix)
        crossover = find_crossover(
            matrix, "Active surveillance", "Thyroid lobectomy"
        )
        for _, row in result.iterrows():
            rows.append(
                {
                    "Scope_scenario": scenario,
                    "Alternative": row["Alternative"],
                    "Baseline_rank": int(row["Rank"]),
                    "Baseline_closeness": row["Closeness_coefficient"],
                    "AS_vs_lobectomy_crossover_guideline_weight": crossover,
                }
            )
    return pd.DataFrame(rows)
```

```
BASELINE_ACTIVE_SURVEILLANCE_PROFILE: Mapping[str, object] = {
    "regional_or_distant_metastases": False,
    "gross_extrathyroidal_extension": False,
    "posterior_progression_threatening_critical_structures": False,
    "primary_growth_mm": 0.0,
    "aggressive_features": False,
```

```

    "reliable_follow_up": True,
    "preference_for_surgery": False,
}

```

```

def active_surveillance_eligibility(profile: Mapping[str, object]) -> tuple[bool, List[str]]:

```

```

    """Evaluate the active-surveillance eligibility predicates.

```

```

    The predicates are guideline-informed author operationalisations of ATA
    Recommendation 14 (Ringel et al., 2025), not literal transcriptions:
    patient anxiety is not modelled; regional and distant metastases are
    merged without the new biopsy-proven qualifier; extrathyroidal extension
    is restricted to gross extension; posterior growth is narrowed to
    progression threatening critical structures; and aggressive cytological
    features are added as an exclusion. Each predicate below is exercised by
    an individual unit test and by combined-failure tests in gate_tests().

```

```

    """

```

```

    reasons: List[str] = []

```

```

    if bool(profile.get("regional_or_distant_metastases", False)):

```

```

        reasons.append("regional or distant metastases")

```

```

    if bool(profile.get("gross_extrathyroidal_extension", False)):

```

```

        reasons.append("gross extrathyroidal extension")

```

```

    if bool(profile.get("posterior_progression_threatening_critical_structures", False)):

```

```

        reasons.append("posterior progression threatening critical structures")

```

```

    if float(profile.get("primary_growth_mm", 0.0)) >= 3.0:

```

```

        reasons.append("primary growth at least 3 mm")

```

```

    if bool(profile.get("aggressive_features", False)):

```

```

        reasons.append("aggressive features")

```

```

    if not bool(profile.get("reliable_follow_up", False)):

```

```

        reasons.append("reliable follow-up unavailable")

```

```

    if bool(profile.get("preference_for_surgery", False)):

```

```

        reasons.append("informed preference for surgery")
    return not reasons, reasons

```

```

def apply_showcase_1_gate(
    profile: Mapping[str, object],
) -> tuple[Dict[str, Sequence[TFN]], pd.DataFrame]:
    eligible, reasons = active_surveillance_eligibility(profile)
    gated = {alternative: tuple(ratings) for alternative, ratings in SHOWCASE_1.items()}
    if not eligible:
        gated.pop("Active surveillance", None)
    log = pd.DataFrame(
        [
            {
                "Alternative": "Active surveillance",
                "Eligible": eligible,
                "Decision": "retained" if eligible else "excluded",
                "Reason": "eligible" if eligible else "; ".join(reasons),
            }
        ]
    )
    return gated, log

```

```

def gate_tests() -> tuple[pd.DataFrame, pd.DataFrame, pd.DataFrame]:
    test_cases = [
        ("Eligible baseline", dict(BASELINE_ACTIVE_SURVEILLANCE_PROFILE), True),
        (
            "Ineligible: unreliable follow-up",
            {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "reliable_follow_up": False},
        )
    ]

```

```

        False,
    ),
    (
        "Boundary below threshold: 2.9 mm growth",
        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "primary_growth_mm": 2.9},
        True,
    ),
    (
        "Boundary at threshold: 3.0 mm growth",
        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "primary_growth_mm": 3.0},
        False,
    ),
    (
        "Ineligible: regional or distant metastases",
        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "regional_or_distant_metastases": True},
        False,
    ),
    (
        "Ineligible: gross extrathyroidal extension",
        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "gross_extrathyroidal_extension": True},
        False,
    ),
    (
        "Ineligible: posterior progression threatening critical structures",
        {
            **BASELINE_ACTIVE_SURVEILLANCE_PROFILE,
            "posterior_progression_threatening_critical_structures": True,
        },
        False,
    ),
    (
        "Ineligible: aggressive features",

```

```

        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "aggressive_features": True},
        False,
    ),
    (
        "Ineligible: informed preference for surgery",
        {**BASELINE_ACTIVE_SURVEILLANCE_PROFILE, "preference_for_surgery": True},
        False,
    ),
    (
        "Combined failure: unreliable follow-up and 3.2 mm growth",
        {
            **BASELINE_ACTIVE_SURVEILLANCE_PROFILE,
            "reliable_follow_up": False,
            "primary_growth_mm": 3.2,
        },
        False,
    ),
    (
        "Combined failure: metastases, gross extension and aggressive features",
        {
            **BASELINE_ACTIVE_SURVEILLANCE_PROFILE,
            "regional_or_distant_metastases": True,
            "gross_extrathyroidal_extension": True,
            "aggressive_features": True,
        },
        False,
    ),
]
test_rows = []
log_rows = []
for name, profile, expected in test_cases:
    eligible, reasons = active_surveillance_eligibility(profile)

```

```

test_rows.append(
    {
        "Test_case": name,
        "Expected_eligible": expected,
        "Observed_eligible": eligible,
        "Pass": eligible == expected,
    }
)
log_rows.append(
    {
        "Test_case": name,
        "Alternative": "Active surveillance",
        "Eligible": eligible,
        "Decision": "retained" if eligible else "excluded",
        "Reason": "eligible" if eligible else "; ".join(reasons),
    }
)

```

```

stress_profile = {
    **BASELINE_ACTIVE_SURVEILLANCE_PROFILE,
    "reliable_follow_up": False,
}
stress_matrix, stress_log = apply_showcase_1_gate(stress_profile)
stress_ranking = fuzzy_topsis(stress_matrix)
stress_ranking.insert(0, "Scenario", "Unreliable follow-up gate stress test")
return pd.DataFrame(test_rows), pd.DataFrame(log_rows), stress_ranking

```

```

def concentration_sensitivity(
    matrix: Mapping[str, Sequence[TFN]],
    showcase: str,

```

```

seed: int,
concentrations: Sequence[float] = (50.0, 100.0, 150.0, 300.0),
iterations: int = 30_000,
) -> pd.DataFrame:
    rows = []
    for concentration in concentrations:
        summary, _, margin = monte_carlo(
            matrix,
            iterations=iterations,
            concentration=concentration,
            seed=seed,
            retain_draws=False,
        )
        mean_margin = float(
            margin.loc[margin["Metric"] == "Mean top-versus-second margin", "Value"].iloc[0]
        )
        for _, row in summary.iterrows():
            rows.append(
                {
                    "Showcase": showcase,
                    "Dirichlet_concentration": concentration,
                    "Alternative": row["Alternative"],
                    "Conditional_top_rank_frequency": row[
                        "Conditional_top_rank_frequency"
                    ],
                    "Mean_closeness": row["Mean_closeness"],
                    "Mean_top_vs_second_margin": mean_margin,
                }
            )
    return pd.DataFrame(rows)

```

```

def input_provenance_table() -> pd.DataFrame:
    """Variable-level separation of sources, assumptions and outputs."""
    rows = [
        (
            "Decision nodes; recommendation direction, strength and certainty; eligibility conditions",
            "2025 ATA guideline",
            "Guideline-derived",
            "Used as reported; no patient-level data",
            "Source-checked",
        ),
        (
            "Typical 100-150 mCi activity for high-risk postoperative RAI",
            "2025 ATA guideline",
            "Guideline-derived",
            "Used to define the illustrative high-risk alternative",
            "Source-checked",
        ),
        (
            "Strength and certainty TFNs",
            "Table 1 numerical anchors",
            "Author-assigned modelling parameter",
            "Transparent ordinal-to-fuzzy encoding",
            "Requires expert calibration",
        ),
        (
            "Applicability and performance-rating TFNs",
            "Table 1 numerical anchors",
            "Author-assigned modelling parameter",
            "Represents exact, partial, poor-match and linguistic ratings",
            "Requires expert calibration",
        ),
    ]

```

```

(
  "Seven baseline criterion weights",
  "0.25, 0.20, 0.15, 0.15, 0.10, 0.10, 0.05",
  "Author-assigned synthetic input",
  "Balanced demonstration; varied in sensitivity analyses",
  "Not clinically validated",
),
(
  "Showcase profiles and contextual ratings",
  "Tables 3-4 and full fuzzy matrices",
  "Author-constructed synthetic input",
  "Scenario logic selected before ranking",
  "Not patient data",
),
(
  "Recommendation 15 partial applicability in the flat parent-node showcase",
  "Scope-qualified partial proxy",
  "Author-assigned scope mapping",
  "Stress-tested against three alternative mappings",
  "Interpret with scope sensitivity",
),
(
  "Complement for a recommendation against an alternative",
  "1 minus the strength-certainty product, then applicability",
  "Author-specified modelling rule",
  "Creates directional guideline-concordance input",
  "Requires external calibration",
),
(
  "Fuzzy TOPSIS distance and closeness",
  "Summed criterion-level Chen vertex distances",
  "Method-derived calculation",

```

```

        "Same aggregation used in deterministic and sampled analyses",
        "Computationally reproduced",
    ),
    (
        "Triangular sampling of TFNs",
        "Bounded draws with modal emphasis",
        "Conditional robustness assumption",
        "Numerical stress test; not an empirical probability model",
        "Interpret as conditional",
    ),
    (
        "Dirichlet weight concentration",
        "50, 100, 150 and 300; primary value 150",
        "Conditional robustness assumption",
        "Controls dispersion around baseline mean weights",
        "Sensitivity-tested",
    ),
    (
        "Ranks, closeness coefficients, margins and rank frequencies",
        "Generated by the supplied script",
        "Calculated output",
        "Properties of the specified synthetic inputs",
        "Reproducible; not clinical effect estimates",
    ),
]
return pd.DataFrame(
    rows,
    columns=["Input", "Value_or_source", "Source_type", "Rationale", "Validation_status"],
)

```

```

def matrix_to_input_table(matrix: Mapping[str, Sequence[TFN]]) -> pd.DataFrame:
    rows = []
    for alternative, ratings in matrix.items():
        row = {"Alternative": alternative}
        for criterion, rating in zip(CRITERIA, ratings):
            row[criterion] = f"({rating[0]:.3f}, {rating[1]:.3f}, {rating[2]:.3f})"
        rows.append(row)
    return pd.DataFrame(rows)

def find_crossover(
    matrix: Mapping[str, Sequence[TFN]], first: str, second: str, steps: int = 10001
) -> float | None:
    """Locate the first true rank crossover of ``first`` versus ``second``.

    A crossover requires a genuine sign change ( $\text{previous} * \text{current} < 0$ ) and is
    reported as the linearly interpolated threshold inside the bracketing grid
    interval. An exact tie on an interior grid point is reported at that point.
    A zero difference occurring only at the endpoint  $\lambda = 1.0$  is a tie at
    the boundary, not a reversal and is reported as ``None`` (callers describe
    this as no reversal below 1.0).

    """
    grid = np.linspace(0.0, 1.0, steps)
    previous = None
    previous_lambda = None
    for lambda_guideline in grid:
        result = fuzzy_topsis(matrix, weights_for_guideline_share(lambda_guideline))
        scores = dict(
            zip(result["Alternative"], result["Closeness_coefficient"], strict=True)
        )
        difference = scores[first] - scores[second]

```

```

if previous is not None:
    if previous * difference < 0.0:
        interpolated = previous_lambda + (0.0 - previous) * (
            lambda_guideline - previous_lambda
        ) / (difference - previous)
        return float(interpolated)
    if difference == 0.0 and lambda_guideline < 1.0:
        return float(lambda_guideline)
    previous = difference
    previous_lambda = lambda_guideline
return None

```

NOTE: The plotting utilities that generate the figures are omitted from this
listing because of space constraints.

```

def main(output_directory: str | Path = "outputs") -> None:
    output_dir = Path(output_directory)
    output_dir.mkdir(parents=True, exist_ok=True)

    baseline_1 = fuzzy_topsis(SHOWCASE_1)
    baseline_2 = fuzzy_topsis(SHOWCASE_2)
    baseline_1.insert(0, "Showcase", "Showcase 1")
    baseline_2.insert(0, "Showcase", "Showcase 2")
    baseline = pd.concat([baseline_1, baseline_2], ignore_index=True)
    baseline.to_csv(output_dir / "baseline_fuzzy_topsis_results.csv", index=False)

    matrix_to_input_table(SHOWCASE_1).to_csv(
        output_dir / "showcase_1_fuzzy_inputs.csv", index=False
    )
    matrix_to_input_table(SHOWCASE_2).to_csv(

```

```

        output_dir / "showcase_2_fuzzy_inputs.csv", index=False
    )

    sensitivity_1 = sensitivity_analysis(SHOWCASE_1, np.linspace(0.0, 0.80, 17))
    sensitivity_2 = sensitivity_analysis(SHOWCASE_2, np.linspace(0.0, 0.50, 11))
    sensitivity_1.to_csv(output_dir / "showcase_1_weight_sensitivity.csv", index=False)
    sensitivity_2.to_csv(output_dir / "showcase_2_weight_sensitivity.csv", index=False)

    mc_1, draws_1, margin_1 = monte_carlo(
        SHOWCASE_1, iterations=30_000, concentration=150.0, seed=202608061
    )
    mc_2, draws_2, margin_2 = monte_carlo(
        SHOWCASE_2, iterations=30_000, concentration=150.0, seed=202608062
    )
    mc_1.insert(0, "Showcase", "Showcase 1")
    mc_2.insert(0, "Showcase", "Showcase 2")
    pd.concat([mc_1, mc_2], ignore_index=True).to_csv(
        output_dir / "monte_carlo_summary.csv", index=False
    )
    draws_1.to_csv(output_dir / "showcase_1_monte_carlo_draws.csv", index=False)
    draws_2.to_csv(output_dir / "showcase_2_monte_carlo_draws.csv", index=False)
    margin_1.insert(0, "Showcase", "Showcase 1")
    margin_2.insert(0, "Showcase", "Showcase 2")
    pd.concat([margin_1, margin_2], ignore_index=True).to_csv(
        output_dir / "monte_carlo_margin_summary.csv", index=False
    )

    concentration_results = pd.concat(
        [
            concentration_sensitivity(
                SHOWCASE_1, "Showcase 1", seed=202608061
            )
        ]
    )

```

```

    ),
    concentration_sensitivity(
        SHOWCASE_2, "Showcase 2", seed=202608062
    ),
],
ignore_index=True,
)

concentration_results.to_csv(
    output_dir / "dirichlet_concentration_sensitivity.csv", index=False
)

recommendation_15_scope_analysis().to_csv(
    output_dir / "recommendation15_scope_sensitivity.csv", index=False
)

input_provenance_table().to_csv(
    output_dir / "input_provenance.csv", index=False
)

gate_test_results, gate_log, gate_stress_ranking = gate_tests()
gate_test_results.to_csv(output_dir / "gate_test_results.csv", index=False)
gate_log.to_csv(output_dir / "eligibility_gate_log.csv", index=False)
gate_stress_ranking.to_csv(
    output_dir / "gate_stress_test_ranking.csv", index=False
)

model_ablation_rows = []
for showcase_name, matrix in (("Showcase 1", SHOWCASE_1), ("Showcase 2", SHOWCASE_2)):
    for model_name, lambda_guideline in (
        ("Context only", 0.0),
        ("Proposed baseline", 0.25),
        ("Guideline only", 1.0),

```

```

):
    result = fuzzy_topsis(matrix, weights_for_guideline_share(lambda_guideline))
    for _, row in result.iterrows():
        model_ablation_rows.append(
            {
                "Showcase": showcase_name,
                "Model": model_name,
                "Guideline_weight": lambda_guideline,
                "Alternative": row["Alternative"],
                "Rank": int(row["Rank"]),
                "Closeness_coefficient": row["Closeness_coefficient"],
            }
        )
    )
pd.DataFrame(model_ablation_rows).to_csv(
    output_dir / "model_ablation_results.csv", index=False
)

crossover = pd.DataFrame(
    {
        "Showcase": ["Showcase 1", "Showcase 2"],
        "Comparison": [
            "Active surveillance versus thyroid lobectomy",
            "Adjuvant RAI versus low-activity RAI",
        ],
        "Approximate_crossover_guideline_weight": [
            find_crossover(SHOWCASE_1, "Active surveillance", "Thyroid lobectomy"),
            find_crossover(
                SHOWCASE_2,
                "Adjuvant RAI (100-150 mCi)",
                "Low-activity RAI (30-50 mCi)",
            ),
        ],
    }
)

```

```

    }
)
crossover.to_csv(output_dir / "crossover_thresholds.csv", index=False)

# NOTE: Figure-generation calls omitted from this listing (see note above).

print("Baseline results")
print(baseline.to_string(index=False))
print("\nConditional robustness simulation summary")
print(pd.concat([mc_1, mc_2], ignore_index=True).to_string(index=False))
print("\nCrossover thresholds")
print(crossover.to_string(index=False))

if __name__ == "__main__":
    import argparse

    parser = argparse.ArgumentParser(
        description=(
            "Run the reproducible Fuzzy-DATA simulations for the two synthetic "
            "differentiated thyroid cancer showcases."
        )
    )
    parser.add_argument(
        "--output-dir",
        type=Path,
        default=Path(__file__).resolve().parent / "outputs",
        help="Directory for generated CSV and PNG files (default: ./outputs).",
    )
    args = parser.parse_args()

```

```
main(args.output_dir)
```

```
EOF
```