ChatGPT

Gradient-Based Output Sensitivity for Finding Language-Specific Neurons in LLMs

Executive summary

Using **gradients of an output-change scalar** with respect to **internal neuron activations** is **theoretically meaningful** and **practically implementable** for identifying language-selective ("language-specific") neurons—*provided you carefully define the scalar objective and validate with interventions*. The core quantity you are proposing, $|\partial s/\partial a_n|$, is exactly an **output sensitivity / local influence** measure, and there is strong prior work showing that gradient-based neuron scoring can approximate the **causal effect of ablation** and can isolate sparse behavior-driving neuron sets. [1]

Three recent strands strongly support the soundness of your direction:

- **PLND** already operationalizes "neuron importance" as a *change in layer output when a neuron is deactivated* (a direct output-change notion), and uses these importance scores to find language-specific neurons and validate them via deactivation experiments. [2]
- **CULNIG** uses a **gradient-based attribution score** $a_n \cdot \partial P/\partial a_n$ and explicitly motivates it as a **first-order (Taylor) approximation** of **ablation effect**, then validates by *masking* the identified neurons. This is extremely close to your proposed idea (and arguably the cleanest precedent). [3]
- **CRANE** argues that activation selectivity can conflate correlation with functional necessity and proposes a relevance-attribution + intervention framework with a dedicated **language-specificity metric (LangSpec-F1)** to quantify targeted degradation under masking. This matches your motivation ("importance, not just activation frequency") and provides an evaluation template. [4]

In practice, the main risks are not conceptual but **methodological**: gradients are **local**, can be **noisy**, and can be confounded by **script/tokenization differences** and by how you define the scalar objective s. The good news is that the literature offers mitigations: **contrastive objectives**, **Integrated Gradients / conductance-style path methods**, smoothing, layerwise normalization, and—most importantly—**intervention sanity checks** (ablation/amplification and targeted masking metrics). [5]

Formalizing sensitivity metrics for language-specific neurons

Setup and notation

Let an autoregressive LLM define a conditional distribution:

$$p_\theta(y \mid x)$$

Let $a_{t,n}^{(\ell)}$ denote the activation of neuron n at token position t in layer ℓ. "Neuron" often means: - an **MLP intermediate dimension** (e.g., SwiGLU gated dimension), as in LAPE's FFN-neuron framing, [6]

- or a **residual stream dimension** (a hidden-state coordinate) $h_{t,d}^{(\ell)}$, which is frequently more scalable (smaller dimensionality) and closer to the logits via the LM head.

Choose a **scalar objective** $s(x, y; \theta)$ (examples in the next section) that represents "output behavior you care about." For a single sample, define the neuron-level sensitivity:

$$g_{t,n}^{(\ell)} = \frac{\partial s}{\partial a_{t,n}^{(\ell)}}$$

You then aggregate across token positions and samples in a language condition L to get a per-neuron score $S_n^{(\ell)}(L)$.

Candidate per-neuron sensitivity metrics

Below are formal definitions aligned with what you requested (raw gradients, Jacobians, IG, conductance, Fisher, contrastive).

Raw gradient magnitude

Token-level:

$$\mathrm{GradMag}_{t,n}^{(\ell)} = \left| \frac{\partial s}{\partial a_{t,n}^{(\ell)}} \right|$$

Sequence aggregation options:

$$\mathrm{GradMag}_{n}^{(\ell)} = \mathrm{Agg}_t \left(\mathrm{GradMag}_{t,n}^{(\ell)} \right)$$

Typical Agg_t: max, mean, or top-k mean.

This is the direct "Jacobian entry magnitude" you described. It is widely used as a sensitivity signal but is known to be noisy and can undercount saturated pathways. [7]

Absolute Jacobian entries $\partial s / \partial a_n$

If you choose s as a scalar derived from the model output (logprob, logit, etc.), then the Jacobian w.r.t. activations is exactly the gradient above; the "Jacobian" framing becomes important when you discuss **vector outputs**. A common practical interpretation is:

$$J_{t,n}^{(\ell)} = \frac{\partial s}{\partial a_{t,n}^{(\ell)}} \quad \Rightarrow \quad |J_{t,n}^{(\ell)}|$$

For a vector output $o \in \mathbb{R}^{|V|}$, you either choose a scalar projection $s = v^\top o$, or consider a norm of J (e.g., Frobenius norm). [8]

Gradient × activation (first-order ablation approximation)

A very strong default (because it approximates "output change if you zero the neuron"):

$$\mathrm{GradAct}^{(\ell)}_{t,n} \;=\; a^{(\ell)}_{t,n} \cdot \frac{\partial s}{\partial a^{(\ell)}_{t,n}}$$

This appears in two highly relevant places:

- CULNIG defines a per-neuron attribution score $s(x, y) = n(x) \cdot \partial P(y \mid x)/\partial n$, and explicitly derives it as a first-order approximation to the causal effect of setting the neuron to 0 via Taylor expansion. [9]
- The neuron-conductance work discusses "gradient*activation" as a baseline and analyzes failure modes of raw activation-only or gradient-only proxies. [10]

In your multilingual setting, this aligns with your intuition: if $|\mathrm{GradAct}|$ differs strongly by language, that suggests differential functional influence.

Integrated gradients on neuron activations

To reduce saturation and improve faithfulness, define a baseline activation a' (e.g., mean activation for a neutral language, or a "masked/zero" activation), and compute:

$$\mathrm{IG}^{(\ell)}_{t,n}(a, a') \;=\; \left(a^{(\ell)}_{t,n} - a'^{(\ell)}_{t,n}\right) \int_{\alpha=0}^{1} \frac{\partial s\left(a' + \alpha(a - a')\right)}{\partial a^{(\ell)}_{t,n}} \, d\alpha$$

This is the internal-unit analogue of Integrated Gradients, introduced with axioms like Sensitivity and Implementation Invariance. [11]
Computationally, it is approximated by m steps along the path; the original IG paper discusses needing on the order of tens to hundreds of steps for good approximation, which matters a lot for LLM-scale work. [12]

Neuron conductance

Conductance can be viewed as "how much IG attribution flows through a hidden unit," and it is formally defined via chain rule decomposition of IG. [13]
For a hidden neuron y, conductance sums over inputs; conceptually for our case, you can interpret it as a path-integrated gradient signal that is better aligned with ablation effects than raw gradients in many settings. [10]

Fisher-based sensitivity scores

The Fisher diagonal is frequently used as a sensitivity/importance measure; it is typically estimated via **squared gradients** of a likelihood objective. [14]
Molchanov et al. connect gradient statistics to the expected Fisher information and interpret gradient-variance/outer-product connections as Fisher-like importance signals. [15]

For neuron activations, a pragmatic "activation-Fisher" analogue is:

$$\mathrm{FisherAct}_{t,n}^{(\ell)} \;=\; \mathbb{E}\left[\left(\frac{\partial \log p_\theta(y \mid x)}{\partial a_{t,n}^{(\ell)}}\right)^2\right]$$

estimated over samples (and optionally over tokens). This measures how strongly small perturbations of a_n change the log-likelihood—an information-theoretic sensitivity.

Contrastive language-specificity scores

Let $S_n(L)$ be your aggregated neuron score under language L. Contrastive "specificity" can be defined as:

- **Difference**:

$$\Delta_n(L; L_0) = S_n(L) - S_n(L_0)$$

- **Ratio**:

$$\rho_n(L; L_0) = \frac{S_n(L) + \epsilon}{S_n(L_0) + \epsilon}$$

- **Z-score across languages**:

$$z_n(L) = \frac{S_n(L) - \mu_n}{\sigma_n}$$

CRANE's central motivation is exactly to move from "language preference" to "functional contribution," and its evaluation metric (LangSpec-F1) operationalizes contrastiveness under interventions. [16]

Summary table of metrics

Metric family	Per-token definition	What it measures	Cost per batch	Notes
Raw gradient $\lvert\partial s/\partial a\rvert$	$\left\lvert\frac{\partial s}{\partial a_{t,n}}\right\rvert$	Local sensitivity of scalar output	1 backward	Can be noisy and miss saturated effects. [17]
Grad×Act	$a_{t,n} \cdot \frac{\partial s}{\partial a_{t,n}}$	First-order approximation of ablation effect	1 backward	Explicitly motivated as ablation approximation in CULNIG. [9]
Integrated gradients	$(a - a') \int_0^1 \partial s/\partial a(\alpha)\, d\alpha$	Path-integrated attribution (mitigates saturation)	m backwards	IG is axiomatized; m can be 20–300+ in practice. [18]
Conductance	IG flow via neuron (chain-rule decomposition)	How attribution passes through hidden unit	m backwards	Designed to fix known issues of activation/grad heuristics. [10]

Metric family	Per-token definition	What it measures	Cost per batch	Notes
Fisher-style	$\mathbb{E}[(\partial \log p / \partial a)^2]$	Expected sensitivity in log-likelihood geometry	Many samples	Fisher diagonal estimated via squared gradients; activation-analogue is natural. [19]
Contrastive	Δ / ρ / z over languages	Language selectivity of sensitivity signal	same as base	CRANE provides a contrastive intervention metric (LangSpec-F1). [20]

Choosing the scalar output objective s

The method is only as "language-specific" as your scalar objective. In multilingual LLMs, two kinds of objectives are common:

- **Capability objectives**: "How much does this neuron affect correct prediction / perplexity in language L?"
- **Language-control objectives**: "How much does this neuron affect whether the output is in language L?"

Both appear implicitly in LAPE's perplexity-based evaluations and language-steering experiments, and explicitly in PLND/CRANE via targeted masking and downstream evaluation. [21]

Practical s choices and tradeoffs

Scalar objective s	Definition (illustrative)	Pros	Cons / confounds
Logprob of gold tokens (teacher forcing)	$s = \sum_t \log p_\theta(y_t \mid x_{<t})$	Directly tied to LM performance; comparable to perplexity-style evaluation used in LAPE. [6]	Tokenization length differs across languages; mixes content + language; needs careful normalization (per token / per character). Tokenization biases are nontrivial across scripts. [22]
Next-token logit / logprob for a target token	$s = \text{logit}(v^*)$ or $\log p(v^*)$	Very targeted; good for "this behavior now" analyses; aligns with neuron-editing style work in factual settings. [23]	High variance; strong dependence on prompt design and tokenization; may not represent "language" rather than "content." [24]

Scalar objective s	Definition (illustrative)	Pros	Cons / confounds
Language-ID score from hidden state	$s = \log p_\phi(L \mid h)$ (probe/classifier)	Cleanly targets "language identity"; helps avoid content confounds	Requires a probe/data; can introduce probe artifacts; must control for script/token features. Script effects are large in practice. [25]
Contrastive language logprob	$s = \log p_\theta(\mathrm{gold}_L \mid x) - \log p_\theta(\mathrm{gold}_{L_0} \mid x)$	Encourages language discrimination; naturally supports contrastive neuron ranking	Needs parallel/paired targets or carefully constructed alternatives; can encode stylistic differences
Layer-output change proxy (module-level)	$s = \|h_{\ell+1} - h_{\ell+1}^{(-n)}\|$	Direct "change in layer output" framing; PLND effectively does this for neuron importances within layers. [26]	Less directly tied to end-task performance; may need downstream validation
Open-ended language accuracy (non-differentiable)	language detector on generated text	Matches "output language control" evaluation used in language-steering work	Not differentiable; must be used only for *evaluation*, not s itself. [27]

Recommended starting choice for your use case

For "language-specific neurons" in the **functional** sense, two choices usually work best:

1. **Teacher-forced gold logprob**, normalized properly (e.g., per-token average), because it directly measures capability and links to LAPE's perplexity evaluation paradigm. [6]
2. A **contrastive scalar** (difference between target-language and non-target baselines) if your goal is specifically to surface differential language influence (this aligns with CRANE's contrastive evaluation philosophy under intervention). [20]

If you want neurons that control *language identity of outputs* (steering), a language-ID score on internal states is cleaner, but you must control for script/tokenization. [25]

Experimental protocol and validation

Dataset design: parallel vs monolingual

Monolingual corpora are simple and are what LAPE uses (Wikipedia corpora; large token budgets per language) for activation-probability statistics. [6]
Parallel data (same meaning across languages) reduces the biggest confound: "neuron reacts to content, not language." PLND also uses curated corpora (e.g., OSCAR language corpora) and evaluates on multilingual tasks like XLSum. [28]

A practical compromise is:

- Use **monolingual corpora** for broad coverage and stable statistics,
- then validate top candidates on a **parallel subset** or carefully aligned tasks (e.g., multilingual MT/QA benchmarks are commonly used in multilingual evaluation setups). [29]

Script/tokenization controls

Script and tokenization effects can mimic "language specificity." Evidence across recent work:

- Non-Latin scripts can show substantial performance gaps and different behavior in multilingual LLMs; this motivates explicit script controls. [30]
- Tokenization and representation biases differ systematically between Latin and non-Latin scripts and correlate with downstream behavior; this is a direct confound for gradient/attribution statistics computed over tokens. [31]
- Large-scale language neuron studies report non-Latin scripts showing greater specialization and lower overlap, which could be real—but could also be inflated by script/tokenization differences if not controlled. [32]

Minimum recommended controls:

- Compare languages with **shared script** (e.g., Romance/Germanic) vs **different script** language pairs. [32]
- Normalize objectives by **tokens vs characters/bytes** to check robustness (tokenization parity issues can strongly change statistics). [31]
- Add **content-matched controls** where possible (parallel sentences or translation of the same prompts). [29]

Per-input computation procedure

A practical per-input pipeline:

1. Run a forward pass, storing target activations $a_{t,n}^{(\ell)}$ via hooks (or internal output capture). [33]
2. Compute scalar s (e.g., mean logprob of gold tokens). [6]
3. Compute gradients $\partial s / \partial a$ using a single backward/`autograd.grad`. [34]
4. Produce per-token scores (e.g., Grad×Act) and aggregate across tokens.

A key empirical point from CULNIG that directly impacts your aggregation choice: they compute token-level attribution and then take the **maximum over token positions**, arguing that not all tokens carry the phenomenon of interest; they also show that mean aggregation can fail to pick behaviorally relevant neurons in their setting. [35]

Aggregation across samples and languages

For each language L, compute per-sample per-neuron scores, then aggregate:

- Use **robust statistics** (median / trimmed mean) to reduce outlier prompts.

- Track **variance**; CULNIG and CRANE both emphasize stability and distributional statistics rather than single-example diagnostics. [36]

For language specificity ranking, prefer **contrastive** aggregation:

- $S_n(L)$ vs $S_n(\mathrm{all} \setminus L)$
- or $z_n(L)$ as language-conditioned z-score across languages.

CRANE's evaluation supports the idea that selectivity should be judged by **targeted degradation vs non-target stability** rather than by selectivity alone. [37]

Intervention sanity checks (non-negotiable)

Because gradients remain correlational/local, validate candidates with interventions:

- **Ablation / masking**: set selected neuron activations to 0 (or a baseline) at inference time and measure performance change. LAPE's key evidence is that deactivating identified neurons degrades the target language more than others. [6]
- **Amplification**: scaling or boosting activations; neuron manipulation studies demonstrate that increasing selected neuron activations can steer model behavior (e.g., language steering or concept behavior), which you can use as a causal probe. [38]
- **Targeted selectivity metric**: adopt CRANE's LangSpec-F1 framing, which operationalizes "language specificity" as targeted drop on target language with minimal non-target drop under the same neuron budget. [39]

Also note a subtle but important negative result: "setting activations to zero" is not always a good proxy for "deactivation," and some interventions may not degrade performance as expected; this is discussed explicitly in follow-up negative-results work. This implies you should test multiple baselines (0, mean, low percentile) rather than relying on 0 only. [40]

Mermaid flowchart for the experiment pipeline

```
flowchart TD
  A[Select languages and model] --> B[Choose dataset: monolingual + parallel
subset]
  B --> C[Define scalar objective s: logprob / contrastive / lang-ID]
  C --> D[Forward pass with hooks: capture a^{(l)}_{t,n}]
  D --> E[Compute s(x,y)]
  E --> F[Compute grads: ∂s/∂a via autograd.grad]
  F --> G[Per-token neuron scores: |∂s/∂a| or a·∂s/∂a]
  G --> H[Aggregate over tokens: max / top-k / mean]
  H --> I[Aggregate over samples per language]
  I --> J[Contrastive scoring across languages: z-score, ratio, Δ]
  J --> K[Candidate neuron sets per language (budgeted top-k)]
  K --> L[Intervention checks: mask/scale candidates]
  L --> M[Evaluate across languages; compute LangSpec-F1-like metric]
  M --> N[Plots & reporting; iterate on s and controls]
```

Scaling, compute, and implementation outline

Computational cost and scaling advice

One-backward methods (raw gradient, Grad×Act) are the most practical at LLM scale: one forward + one backward per batch. This is exactly why CULNIG motivates its score as a single-run approximation rather than brute-force ablation per neuron, which would be infeasible for millions of neurons. [41]

Path methods (Integrated Gradients, conductance) multiply cost by m interpolation steps. The IG paper describes Riemann-sum approximation and discusses step counts that can reach dozens to hundreds, which is often too slow for large multilingual sweeps. [42]

Relevance propagation alternatives: CRANE uses LRP/AttnLRP to score neurons by output-level relevance rather than activation magnitude. AttnLRP is presented as enabling faithful transformer attributions with efficiency comparable to a single backward pass and supporting latent attributions. This provides a strong baseline or complement to gradient methods if compute allows. [43]

Activation storage is the real bottleneck. To scale on LLaMA/Mistral-size models, prefer:

- Score only a few layers (bottom/top layers are often implicated in language neuron work). [44]
- Score only **MLP neurons** (or only gate projection neurons) rather than everything. PLND and follow-ups explicitly treat FFN/attention structure as separable units. [45]
- Or score **residual stream dimensions** first (smaller than MLP intermediate width), then drill down with targeted MLP scoring.

Memory-saving tactics you can use immediately:

- **Gradient checkpointing**: trades compute for memory by rerunning forward segments during backward. [46]
- Minimize stored tensors: hook only what you need; avoid `output_hidden_states=True` for all layers unless necessary.
- Use smaller sequence lengths or sample token positions (e.g., focus on last token or linguistically salient positions).

A concrete empirical reference point: CULNIG reports that computing neuron attribution scores across full models can take hours per model even on high-end GPUs (they report multi-hour scoring runs). This highlights the need for sampling and layer selection in your setting. [47]

Low-cost approximations and variants

These approximations preserve your core idea while reducing cost:

- **Top-k token positions**: mirror CULNIG's "max over tokens" rationale (only some tokens express the phenomenon), but implement "top-k max" to reduce noise. [48]
- **Subset neurons by prefilter**: use LAPE-like activation filters to narrow candidate neurons, then apply your gradient sensitivity only on that subset (two-stage: cheap → precise). [44]

- **Layerwise scoring then drilldown**: first compute per-layer norms of $\partial s/\partial h^{(\ell)}$ (residual stream), and only for high-signal layers compute per-MLP-neuron scores.
- **Contrastive pairs only**: for each language, compare against one reference language (e.g., English) rather than all languages; PLND and CRANE both emphasize asymmetries involving English-centric processing. [49]
- **LoRA-style targeted adaptation for validation**: negative-results follow-ups explicitly study neuron-level fine-tuning and show it may not yield cross-lingual transfer gains; still, small targeted adaptation can be used as a *diagnostic* rather than a goal. [50]

PyTorch and Hugging Face implementation outline

This sketch shows how to compute per-neuron sensitivity for:

- last-layer **MLP intermediate neurons** (SwiGLU gated vector), and
- last-layer **residual stream dimensions**.

It uses `torch.autograd.grad` for gradient computation. [51]
It also assumes a Transformers-style causal LM; gradient checkpointing can be enabled via Transformers utilities if desired. [52]

```
import torch
import torch.nn.functional as F
from transformers import AutoModelForCausalLM, AutoTokenizer

# ----------------------------
# Utilities: scalar objectives
# ----------------------------
def teacher_forced_mean_logprob(logits, labels, attention_mask=None):
    """
    logits: [B, T, V]
    labels: [B, T] with -100 for ignored positions
    returns scalar s = mean logprob over non-ignored tokens
    """
    logp = F.log_softmax(logits, dim=-1)  # [B, T, V]
    # Gather logp at gold labels
    valid = labels != -100
    gold = torch.zeros_like(labels, dtype=logp.dtype)
    gold[valid] = logp[valid, labels[valid]]
    denom = valid.sum().clamp_min(1)
    return gold[valid].sum() / denom

def contrastive_logprob(s_target, s_ref):
    return s_target - s_ref

# ----------------------------
# Hook helpers
# ----------------------------
```

```python
class ActivationCache:
    def __init__(self):
        self.tensors = {}

    def save(self, name, tensor):
        self.tensors[name] = tensor
        tensor.retain_grad()  # so grad is accessible after backward/grad call

def find_last_layer_modules(model):
    """
    Heuristic for decoder-only TransformerLMs (LLaMA/Mistral-like).
    You will likely need to adapt names depending on architecture:
      - model.model.layers[-1] is typical for LLaMA-like
      - MLP module often at layer.mlp
      - MLP projections often gate_proj, up_proj, down_proj
    """
    last = model.model.layers[-1]
    return last, last.mlp

# ----------------------------
# Core: compute neuron sensitivity
# ----------------------------
@torch.no_grad()
def describe_tensor(t):
    return {"shape": tuple(t.shape), "dtype": str(t.dtype), "device":
str(t.device)}

def compute_sensitivity(
    model,
    tokenizer,
    texts,
    *,
    max_length=256,
    device="cuda",
    target="mlp_intermediate",  # or "residual"
    token_agg="max",            # "max" or "mean" or "topk"
):
    model.eval()
    model.to(device)

    batch = tokenizer(
        texts,
        return_tensors="pt",
        padding=True,
        truncation=True,
        max_length=max_length,
    ).to(device)
```

```
    input_ids = batch["input_ids"]
    attention_mask = batch.get("attention_mask", None)

    # Teacher forcing: predict next token
    labels = input_ids.clone()
    labels[:, :-1] = input_ids[:, 1:]
    labels[:, -1] = -100  # ignore last (no next token)
    if attention_mask is not None:
        # Ignore padding positions in loss/logprob
        labels = labels.masked_fill(attention_mask == 0, -100)

    cache = ActivationCache()
    last_layer, last_mlp = find_last_layer_modules(model)

    hooks = []

    if target == "residual":
        # Capture residual stream (hidden states) entering or leaving last
block.
        def resid_hook(module, inputs, output):
            # output is typically the hidden states after the block
            cache.save("residual_out", output)
        hooks.append(last_layer.register_forward_hook(resid_hook))

    elif target == "mlp_intermediate":
        # For SwiGLU MLP, capture the intermediate "gated" vector
        # a = act(gate_proj(x)) * up_proj(x)
        # WARNING: calling projections inside hook doubles compute; for a
production
        # implementation, attach hooks to gate_proj and up_proj to reuse
outputs.
        def mlp_hook(module, inputs, output):
            x = inputs[0]  # [B, T, d_model]
            gate = module.gate_proj(x)
            up = module.up_proj(x)
            a = module.act_fn(gate) * up  # [B, T, d_ff]
            cache.save("mlp_intermediate", a)
        hooks.append(last_mlp.register_forward_hook(mlp_hook))
    else:
        raise ValueError("target must be 'residual' or 'mlp_intermediate'")

    # ---- Forward (with grad enabled) ----
    for h in hooks:
        pass

    # Enable grad for the forward pass because we need ∂s/∂a
    with torch.enable_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask,
```

```
use_cache=False)
        logits = outputs.logits  # [B, T, V]
        s = teacher_forced_mean_logprob(logits, labels,
attention_mask=attention_mask)

        # Choose activation tensor
        if target == "residual":
            a = cache.tensors["residual_out"]
        else:
            a = cache.tensors["mlp_intermediate"]

        # Compute gradient of scalar w.r.t activation tensor
        (grad_a,) = torch.autograd.grad(s, a, retain_graph=False,
create_graph=False)

        # Per-neuron token-level scores
        # Grad×Act (recommended default):
        score_tok = (a * grad_a).abs()  # [B, T, N]

        # Aggregate over tokens -> [B, N]
        if token_agg == "max":
            score_seq = score_tok.max(dim=1).values
        elif token_agg == "mean":
            score_seq = score_tok.mean(dim=1)
        else:
            raise ValueError("token_agg must be 'max' or 'mean'")

        # Aggregate across batch -> [N]
        score = score_seq.mean(dim=0)

    # Cleanup hooks
    for h in hooks:
        h.remove()

    return {
        "scalar_s": float(s.detach().cpu()),
        "activation_info": describe_tensor(a.detach()),
        "score": score.detach().cpu(),  # per-neuron sensitivities
    }

# ----------------------------
# Intervention: ablation / amplification
# ----------------------------
def run_with_neuron_mask(model, layer_idx, neuron_indices, scale=0.0):
    """
    Returns a context manager-like handle that applies a scaling mask
    to selected neurons in the MLP intermediate of a specified layer.
    """
```

```
    layer = model.model.layers[layer_idx]
    mlp = layer.mlp
    neuron_indices = torch.tensor(neuron_indices,
device=next(model.parameters()).device)

    def mlp_hook(module, inputs, output):
        x = inputs[0]
        gate = module.gate_proj(x)
        up = module.up_proj(x)
        a = module.act_fn(gate) * up
        a[..., neuron_indices] = scale * a[..., neuron_indices]
        # Recompute output with modified a
        out = module.down_proj(a)
        return out

    handle = mlp.register_forward_hook(lambda m, inp, out: mlp_hook(m, inp,
out))
    return handle
```

Key implementation notes (based on the literature and the framework constraints above):

- Prefer **Grad×Act** first: it has explicit "first-order ablation effect" justification in CULNIG and aligns with your stated intent ("large change in outputs if neuron changes"). [8]
- Prefer **max-over-tokens** (or top-k max) when the property is expressed sparsely across tokens; CULNIG shows mean aggregation can fail in practice. [53]
- If you later move to IG/conductance, expect a large compute multiplier due to multi-step integration. [54]

Pitfalls, confounds, and mitigation strategies

Gradient locality vs functional necessity

Gradients measure **local sensitivity**, not guaranteed causal necessity. CRANE's central critique is that activation-based heuristics conflate preference with importance and emphasizes interventions to establish functional necessity. [55]
Mitigation: always pair gradient ranking with **masking/ablation** and quantify selectivity with a LangSpec-F1-like metric. [20]

Saturation and attribution failure modes

Raw gradients can be small even when a feature is important (saturation). IG was proposed specifically to address failures of naive sensitivity maps and is grounded in axioms; conductance further addresses hidden-unit attribution quirks. [56]
Mitigation ladder: 1. Grad×Act (cheap, often stronger than raw gradient), [8]
2. SmoothGrad-style averaging to reduce noise, [57]
3. IG / conductance if budget permits. [54]

LayerNorm / scaling confounds

Gradients are scale-sensitive: layernorm and residual pathways can change gradient magnitudes without changing causal role. CRANE addresses cross-language comparability with per-layer normalization and distributional statistics (e.g., kurtosis-based contrasts). [58]
Mitigation: normalize scores within layer (e.g., divide by median absolute deviation per layer) and report per-layer distributions.

Token frequency and script/tokenization confounds

Tokenization differences across scripts can change: - number of tokens,
- token frequency distributions,
- and apparent neuron "importance" if computed per token.

Empirical work documents systematic tokenization/representation bias across Latin vs non-Latin scripts. [59]
Mitigation: - Compare per-token vs per-character normalizations. [60]
- Use script-matched controls and parallel content where possible. [61]
- Include controls designed to filter "superficial" token responders; CULNIG constructs control datasets precisely to subtract non-target mechanisms (e.g., task understanding, entity-name tokens). [62]

Intervention baseline choice (0 is not always "off")

Follow-up work highlights that setting activations to zero may not degrade performance as expected and argues that different baselines (low percentile, etc.) can behave differently; this matters for your ablation validation. [40]
Mitigation: test multiple baselines (0, mean, low-percentile) and report robustness.

Gradient noise and stability

Gradient-based scores can be high-variance. SmoothGrad explicitly proposes averaging gradients under noise to reduce visual/statistical noise. [57]
Mitigation: average across prompts; apply gradient smoothing; use robust aggregations and confidence intervals.

Prior work map and final recommendations

Prior work to cite, with relevance notes

Work	One-line relevance to your idea
Language-Specific Neurons (LAPE)	Identifies language-specific FFN neurons via activation probability entropy; validates with deactivation/steering and perplexity/generation changes—strong baseline to compare your gradient scores against. [63]

Work	One-line relevance to your idea
How do LLMs Handle Multilingualism? (PLND)	Defines neuron "importance" via output change when a neuron is deactivated (layer-output difference norms) and uses it to detect language-specific neurons and validate by targeted deactivation in multilingual tasks. [2]
CRANE	Explicitly argues correlation-based language neuron identification is insufficient; uses relevance attribution (LRP/AttnLRP) and interventions; introduces LangSpec-F1 to quantify targeted degradation—excellent evaluation template for gradient methods. [4]
CULNIG (culture neurons)	Uses $a \cdot \partial P / \partial a$ as a neuron attribution score and derives it as a first-order approximation to ablation effect; validates by masking; provides a near-drop-in methodological blueprint for language specificity. [64]
Integrated Gradients	Provides axiomatized path-integrated attribution that mitigates saturation; can be applied to internal activations for more faithful sensitivity than raw gradients, but is expensive. [65]
How Important Is a Neuron? (conductance)	Defines neuron conductance as IG flow through hidden units; analyzes failure modes of activation/gradient heuristics and links to ablation evaluation. [66]
Knowledge Neurons	Demonstrates neuron-level attribution + suppression/amplification causing large probability changes, supporting neuron-level influence measures and the need for causal interventions. [67]
Fishers for Free? / Fisher-diagonal literature	Formalizes Fisher diagonal as squared-gradient-based sensitivity measure; provides grounding for Fisher-style activation sensitivity variants. [68]
Tokenization/script bias papers	Show systematic script/tokenization disparities that can confound "language specificity" signals, motivating explicit controls. [69]
Negative-result follow-up on language neurons	Cautions that some neuron interventions and "zeroing" choices may not yield expected degradations or transfer gains, underscoring the need for robust intervention design. [70]

Final recommendations

Your idea is worth pursuing, and there is substantial evidence that it can work—especially if you frame it as identifying **functionally language-selective** neurons rather than merely **language-correlated** neurons.

Best first metric to implement - Start with **Grad×Act** on internal activations:

$$a_n \cdot \frac{\partial s}{\partial a_n}$$

because it has explicit theoretical justification as a first-order approximation to ablation effect and has been validated in a closely analogous "specific neurons" setting (CULNIG). [3]

Best first scalar objective - Start with **teacher-forced mean logprob** of gold tokens (capability-based), and add a **contrastive** variant (difference vs a reference language) for ranking. This aligns with how LAPE evaluates multilingual capability (perplexity/language modeling) and with CRANE's contrastive intervention evaluation philosophy. [71]

Minimum validation bar - Adopt a CRANE-like notion of language specificity as **targeted degradation under masking with minimal non-target degradation**, and report an intervention metric (LangSpec-F1 or a simplified version). [39]

Next experiments that most reduce uncertainty - Compare rankings from **LAPE vs PLND vs your gradient score** and measure which ranking best predicts targeted degradation under ablation at fixed neuron budgets. [72]
- Add **control datasets** (script controls, content controls, "task-only" controls) in the spirit of CULNIG to ensure your gradients are not simply tracking superficial tokens or formatting. [73]
- Verify robustness to intervention baseline (0 vs mean vs percentile), since zeroing can behave unexpectedly. [40]

If your end-goal is "language-specific neurons" as *causal components*, the most defensible position—supported directly by CRANE—is: **use gradient-based sensitivity to propose candidates, and intervention metrics to define success.** [74]

---

[1] [3] [8] [9] [35] [41] [48] [62] [64] [73] https://openreview.net/pdf/e85452e248f39684b16124de0ad23e902e6cdfd0.pdf
https://openreview.net/pdf/e85452e248f39684b16124de0ad23e902e6cdfd0.pdf

[2] [26] [28] [45] [49] https://proceedings.neurips.cc/paper_files/paper/2024/file/1bd359b32ab8b2a6bbafa1ed2856cf40-Paper-Conference.pdf
https://proceedings.neurips.cc/paper_files/paper/2024/file/1bd359b32ab8b2a6bbafa1ed2856cf40-Paper-Conference.pdf

[4] [16] [20] [36] [37] [39] [43] [55] [58] [74] CRANE: Causal Relevance Analysis of Language-Specific Neurons in Multilingual Large Language Models
https://arxiv.org/html/2601.04664v1

[5] [11] [12] [18] [42] [54] [56] [65] https://proceedings.mlr.press/v70/sundararajan17a/sundararajan17a.pdf
https://proceedings.mlr.press/v70/sundararajan17a/sundararajan17a.pdf

[6] [21] [27] [38] [44] [63] [71] [72] Language-Specific Neurons: The Key to Multilingual Capabilities in Large Language Models
https://arxiv.org/html/2402.16438v2

[7] [10] [13] [66] https://arxiv.org/pdf/1805.12233
https://arxiv.org/pdf/1805.12233

[14] [19] [68] arxiv.org
https://arxiv.org/pdf/2507.18807

[15] jankautz.com
https://jankautz.com/publications/Importance4NNPruning_CVPR19.pdf

17 24 57 https://arxiv.org/abs/1706.03825
https://arxiv.org/abs/1706.03825

22 31 59 60 https://arxiv.org/pdf/2509.20045
https://arxiv.org/pdf/2509.20045

23 67 aclanthology.org
https://aclanthology.org/2022.acl-long.581.pdf

25 29 30 61 69 https://aclanthology.org/2025.naacl-long.599.pdf
https://aclanthology.org/2025.naacl-long.599.pdf

32 https://arxiv.org/html/2507.22608v1
https://arxiv.org/html/2507.22608v1

33 https://docs.pytorch.org/docs/stable/autograd.html
https://docs.pytorch.org/docs/stable/autograd.html

34 51 https://docs.pytorch.org/docs/stable/generated/torch.autograd.grad.html
https://docs.pytorch.org/docs/stable/generated/torch.autograd.grad.html

40 50 70 https://aclanthology.org/2025.insights-1.6.pdf
https://aclanthology.org/2025.insights-1.6.pdf

46 https://docs.pytorch.org/docs/stable/checkpoint.html
https://docs.pytorch.org/docs/stable/checkpoint.html

47 53 Neuron-Level Analysis of Cultural Understanding in Large Language Models
https://arxiv.org/html/2510.08284v1

52 https://huggingface.co/docs/transformers/en/internal/modeling_utils
https://huggingface.co/docs/transformers/en/internal/modeling_utils