Prompt-Conditioned Semantic Segmentation for Drywall Defect Detection

---

Note : LLMs and CLI tools were used for work on this assignment, but only to the extent of use as a tool, thinking and planning was not outsourced

Index

1. Methodology

1.1 The problem we are trying to solve

The goal of this project is to explore whether **prompt-conditioned segmentation models** can be used to identify drywall defects such as cracks and joint tape using natural language prompts instead of fixed class labels.

In real terms, this means asking a model to segment *“drywall crack”* or *“drywall joint tape”* in an image, and expecting it to highlight only the relevant regions. This is a harder problem than standard semantic segmentation for two main reasons:

1. The defects themselves are visually subtle. Cracks are thin, low-contrast, and often resemble background texture.
2. The model must correctly interpret a **text prompt** and align it with visual features in the image.

Because of this, both vision quality and language-vision alignment matter.

1.2 Why CLIPSeg

This section is designed to serve as a window for you to peek into the way I reason and deliberate on problems and as such is detailed.

I chose **CLIPSeg** because it offers a great middle ground for this problem. CLIPSeg is a text-conditioned segmentation model built on top of CLIP’s joint vision language embeddings, which means segmentation behavior can be steered directly using natural language prompts.

From an engineering perspective, this gives us a few strong advantages:

- First, **class definitions are not hard-coded into the model**. New defect types can be introduced, existing ones refined, or experimental categories tested simply by changing the text prompt. This avoids retraining or re-architecting the model every time the label space changes.

- Second, CLIPSeg allows us to **reuse a single model across multiple defect types** (cracks, joint tape, seams) instead of maintaining one segmentation model per class. This significantly reduces operational complexity, and is that much more efficient.

- Third, CLIPSeg makes it easy to **evaluate zero-shot and low-shot behavior in a controlled way**. This is valuable both for understanding the limits of pre-trained vision language models and for measuring how much domain specific fine tuning is actually necessary to reach acceptable performance.

Trade-offs and Limitations

These benefits come with clear trade-offs :

CLIPSeg is **not optimized for very fine-grained, thin structures**.. It performs better on **large, visually distinct regions** and struggles with **thin cracks, hairline seams, and subtle texture variations**, which are common in drywall imagery.

In contrast, a more traditional alternative such as a **fully supervised CNN-based segmentation model (e.g., U-Net, DeepLab)** would likely achieve better boundary precision and finer localization, especially for cracks.

However even with the given constraints ultimately CLIPSeg was the best choice, because of the simple fact that the CNN based models would not be nearly as **scalable to new labels** and would never generalise.
The way I see it, this assignment is a part of a bigger picture. Origin solves automation for construction and tapes and cracks are a **small subset of a big big superset of construction defects** and as such it is never feasible to train a CNN model that would need frequent re-training, the vision stack at Origin must scale to new unseen defects fairly quickly. This was the main reason for going with CLIPSeg. Another irresistible advantage with CLIPSeg was the ability to utilise human language as a control signal.
All of these advantages and i havent even started my rant on how big of a pain it is creating annotated dataset for CNN models.
Hence I traded some pixel-level precision in exchange for flexibility, prompt-driven control, and a unified modeling approach.

1.3 Model variants evaluated

To understand the core crux and the intricacies associated with the problem we utilise three progressively more advanced and broader reaching techniques, starting with a classic Baseline.

1. **Baseline: Zero-shot CLIPSeg (phase-2)**

   The baseline uses a pretrained CLIPSeg model with a single prompt per class and no task-specific fine tuning.
   This setup was to check the limits of the performance without laying a finger at the weights.. It also serves as a reference point for identifying failure modes before adding complexity.

2. **Ensemble CLIPSeg (phase-3A)**

   In the ensemble setup, multiple prompts are used for the same class, and their outputs are combined using simple aggregation strategies such as **mean** and **max**. The motivation here is straightforward: CLIP models can be sensitive to wording. By averaging across prompts, we aim to reduce that sensitivity and improve robustness.

   This approach is still lightweight and does not require retraining, but it increases inference cost and adds a little post-processing complexity.

3. **Finetuned CLIPSeg (phase-3B)**

   In the fine-tuned setup, CLIPSeg is trained further on drywall-specific data.
   The architecture remains unchanged; only the weights are updated. The goal is not to turn CLIPSeg into a fully supervised segmentation model, but to gently adapt it to:

   - drywall textures
   - defect shapes
   - dataset-specific visual statistics

This setup tests whether modest domain adaptation can improve localization without losing the benefits of prompt conditioning.

1.4 Inference and evaluation flow

Across all variants, the inference pipeline follows the same high-level steps:

1. Load the image
2. Apply the text prompt(s)
3. Generate a probability mask
4. Convert it to a binary mask via thresholding
5. Compare it with the ground truth mask

I use **IoU** and **Dice coefficients** as evaluation metrics as asked in the document outlining the problem statement, and we rely heavily on visual inspection to understand where and why the model fails.

1.5 Technical details and runtime metrics

1.Training Details:

- Architecture: CLIPSeg (CIDAS/clipseg-rd64-refined)
- Training strategy: Head-only fine-tuning (CLIP encoders frozen)
- Loss function: BCE + Dice (strong labels), BCE only (weak labels, 0.5x weight)
- Optimizer: AdamW (lr=1e-4, weight_decay=1e-4)
- Hardware: RTX 4090, FP16 mixed precision
- Early stopping: Best model at epoch 5 (validation IoU: 0.4748)

2.Runtime metrics:

Metrics	**Value**
Training Time	**~45 min (6 epochs, RTX 4090)**
Hardware	**RTX 4090 24GB, FP16 precision**
Batch Size	**16 (training), 1 (inference)**

2. Data Preparation

2.1 Dataset structure

The data pipeline is organized to clearly separate **raw datasets**, **processed training artifacts**, and **final model-ready inputs**. Rather than flattening everything into a single directory, the structure reflects the stages through which the data passes.

At a high level, the data directory contains four conceptual layers:

1. **Raw task-specific images and masks**
2. **Original dataset exports (as provided by the source)**
3. **Processed, model-ready splits**
4. **Task-aligned convenience views for inspection and debugging**

This separation was intentional and helps avoid accidental leakage between training, validation, and evaluation.

2.2 The issue

For several images particularly in the crack and taping datasets **multiple ground-truth masks exist for the same base image**. These masks differ only by a hash-like suffix in the filename.
As a result:

- One image may have multiple valid annotations
- Inference, however, produces exactly one prediction per image

This is a serious issue as you can't do a straightforward one-to-one comparison when an image might have 2-3 ground-truth masks but only produces a single prediction.

2.3 Its significance

This mismatch caused two concrete problems:

1. **Metric computation issues**
   When matching predictions to ground truths by filename, some predictions had no matching mask, while others had multiple possible matches. This led to missing entries and unstable averages (including NaN values in some cases).
2. **Broken visualizations**
   Automated visualization scripts assumed exactly one ground-truth mask per image.

3. When that assumption failed, the scripts either selected an arbitrary mask or failed altogether.

2.4 The fix

Rather than forcing a fragile workaround, the pipeline was adjusted to explicitly acknowledge this limitation:

- Filename normalization and sanity checks were added
- Evaluation was restricted to valid, well-matched pairs
- Visualization generation was split into **separate scripts** for:
    1. baseline
    2. ensemble
    3. finetuned models

This is why the baseline outputs looks soo much cleaner. I built that pipeline first, before discovering these complications. Once I understood what was going on with the dataset, I updated the finetuned and ensemble pipelines to match.

3. Results

3.1 Quantitative results

Quantitative metrics are reported only for image mask pairs that can be matched reliably. Overall trends are consistent across classes:

- The baseline provides a reasonable zero-shot starting point but often over-segments
- The ensemble improves stability and prompt robustness
- Finetuning produces the best overall alignment with ground truth

That said, absolute IoU and Dice scores remain modest, especially for cracks. This reflects the intrinsic difficulty of the task rather than an implementation error.

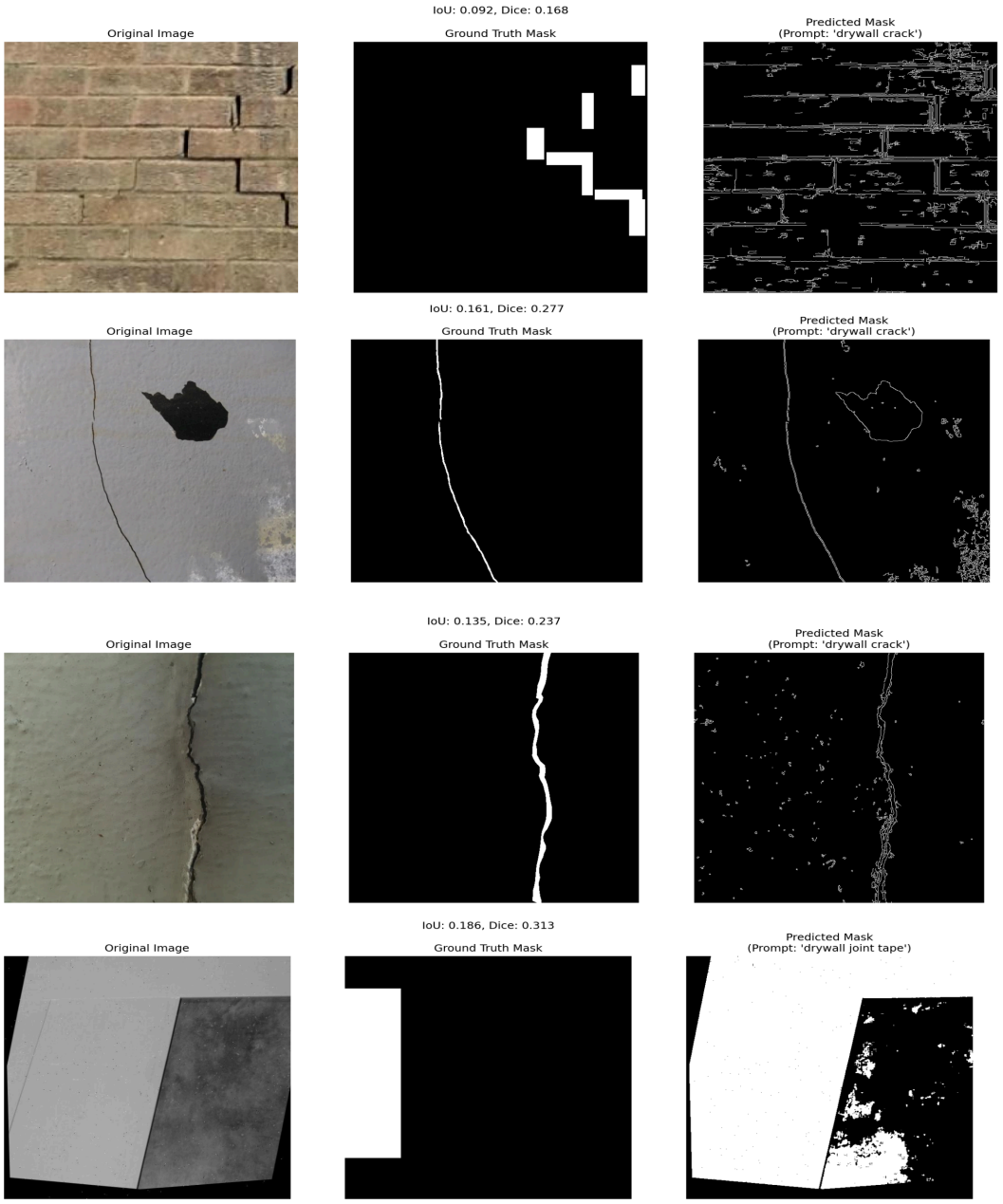
Scores for fine tuning

Method	Dataset	mIoU	Dice	Sample
Baseline	**Cracks**	**0.1430**	**0.2470**	**201**
Baseline	**Taping**	**0.1130**	**0.1840**	**86**
Ensemble	**Cracks**	**0.023(max),0.023(mean)**	**0.044(max),0.044(mean)**	**201**
Ensemble	**Taping**	**0.198(max),0.199(mean)**	**0.312(max),0.314(mean)**	**86**
Finetuned	**Cracks**	**0.4902**	**0.6338**	**201**
Finetuned	**Taping**	**0.4389**	**0.5927**	**86**

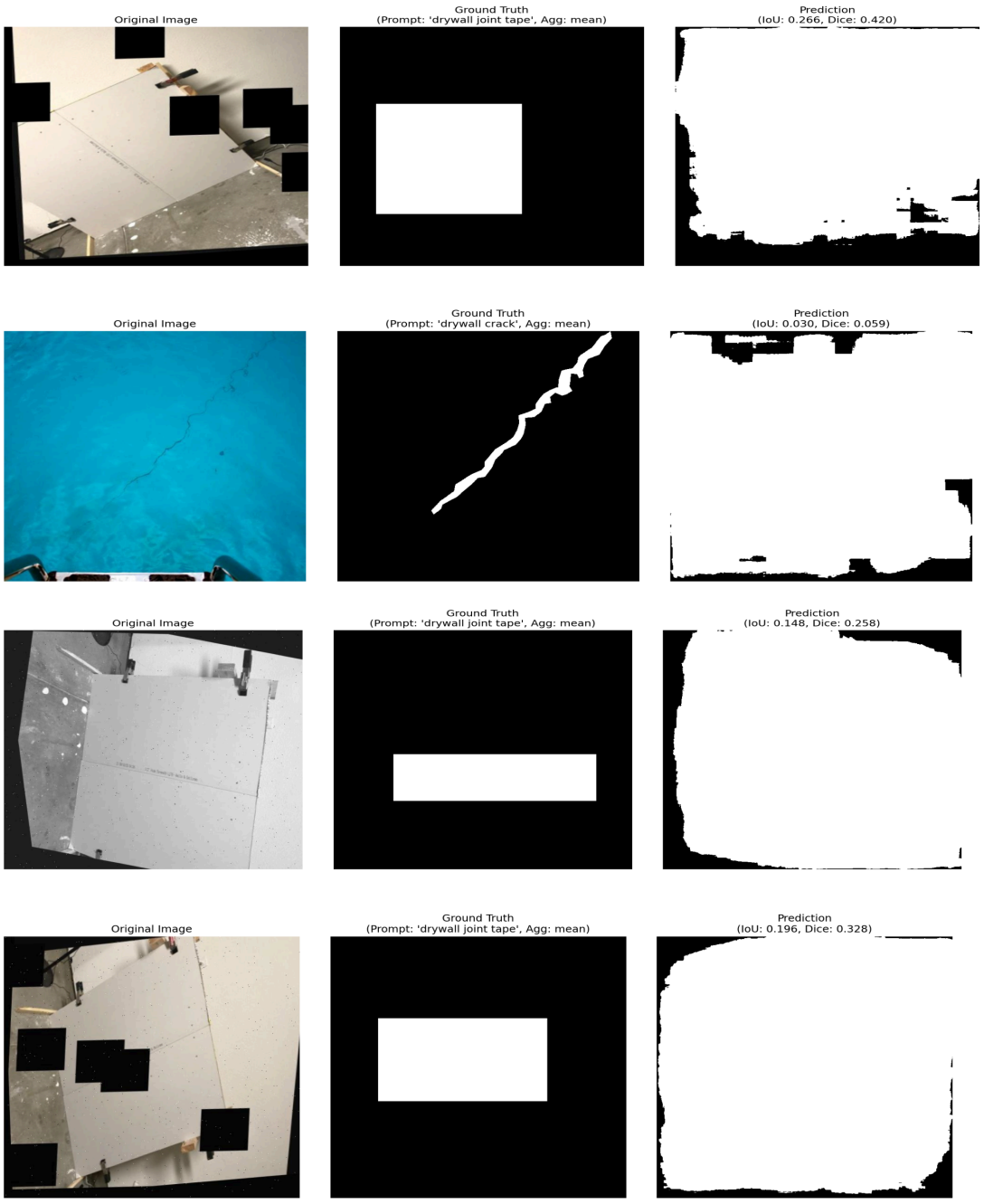
3.2 Qualitative observations

Visual inspection reveals patterns that metrics alone do not capture:

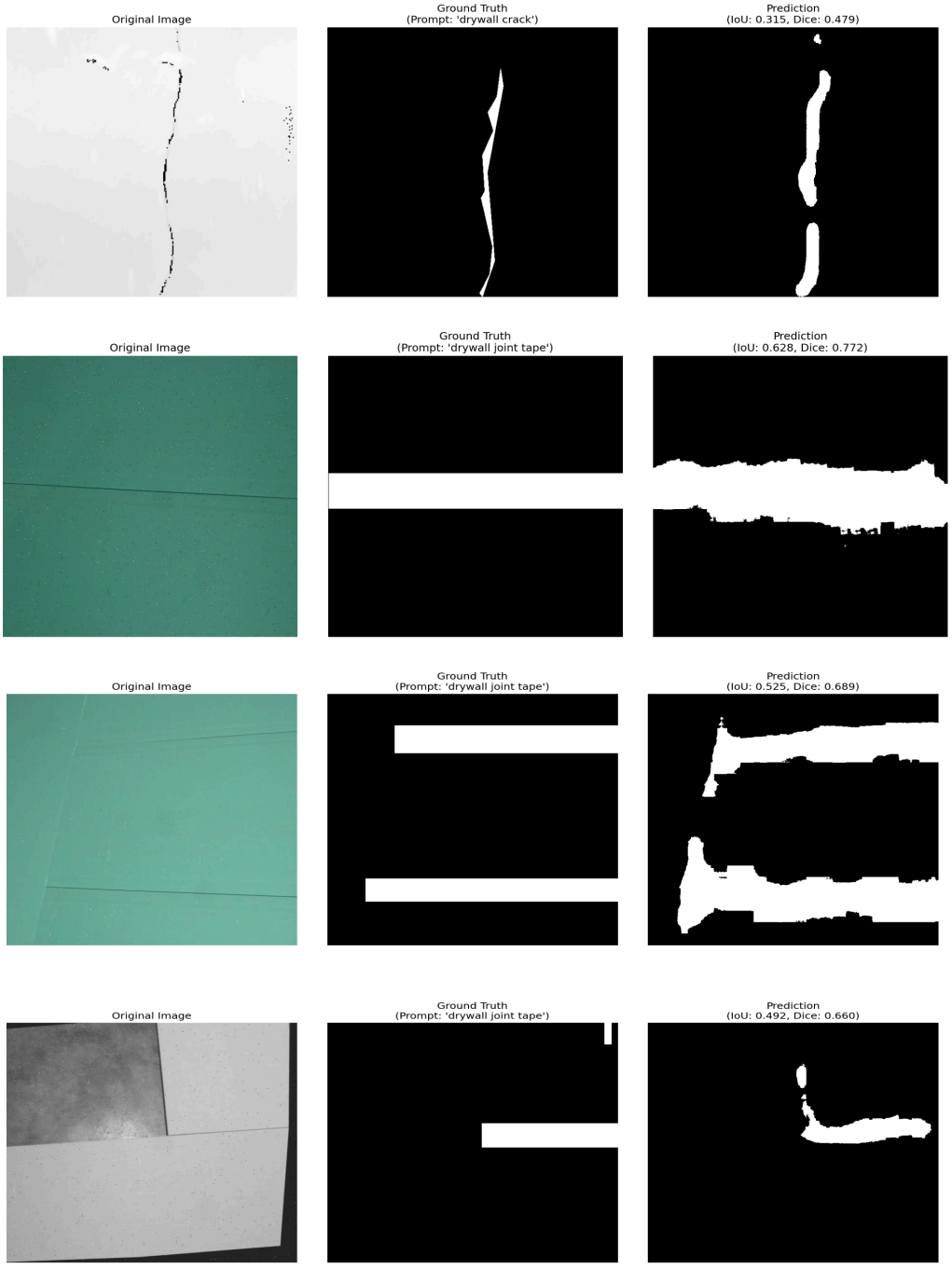
- Baseline predictions often cover large drywall regions instead of isolating defects



- Ensemble predictions are more consistent but still struggle with thin structures



- Finetuned predictions are better localized but can miss extremely subtle cracks



4. Failure Cases and Potential Improvements

As we saw in our quantitative results the fine-tuned model achieves higher quantitative performance on cracks than on taping,however this does not mean that cracks are an easier problem for the said model. In practice, the two classes fail in different ways for different reasons, which we shall explore.

4.1 Background Over-Segmentation

A common failure mode across both datasets is over-segmentation of background regions.

The model often identifies the correct semantic region (e.g., the general area of a crack or joint), but includes large surrounding areas that are not part of the defect. This is especially visible in scenes with:

- Low texture contrast
- Shadows or uneven lighting
- Large homogeneous wall surfaces

This behavior is consistent with CLIPSeg's design: the model prioritizes **semantic relevance** over precise boundary localization.
Potential improvements:

- Morphological post-processing to clean small disconnected regions
- Boundary-aware or edge-focused loss terms
- Multi-scale feature aggregation to better capture fine boundaries

---

4.2 Crack Segmentation: Structurally Hard, Statistically Strong

Cracks remain the most structurally challenging class due to:

- Extremely thin and irregular geometry
- High visual similarity to surface texture
- Annotation inconsistency across images
- Multiple masks per image for the same base image

Despite these challenges, cracks achieve higher mIoU and Dice scores than taping. This is primarily because:

- Crack annotations are polygon-based (strong supervision)
- The crack dataset has more validation samples (201 vs 86)
- The loss function includes Dice loss, which favors thin structures

In other words, cracks perform better **because the supervision is better**, not because the task is simpler.

Potential improvements:

- Consolidating duplicate crack annotations into unified masks
- Using soft or distance-transform supervision to reduce sensitivity to exact boundaries
- Complementing region-based metrics with boundary-based evaluation

---

4.3 Taping Segmentation: Easier Geometry, Weaker Labels

Taping regions are visually larger and more regular, but performance is lower due to:

- Weak supervision from bounding-box–derived masks
- Coarse rectangular annotations that do not reflect true boundaries
- Fewer validation samples (86 images)

These limitations cause the model to learn approximate localization rather than precise segmentation, which is reflected in the lower metrics.

Potential improvements:

- Refining box-derived masks using classical CV heuristics
- Introducing limited manual polygon annotations for calibration
- Adjusting loss weighting dynamically based on annotation quality

---

4.4 Prompt Sensitivity

Performance varies noticeably with prompt phrasing, especially in zero-shot and ensemble settings. Semantically similar prompts can lead the model to attend to different visual cues. This is expected in text-conditioned models and highlights a key tradeoff of language-driven segmentation.

Potential improvements:

- Prompt learning or prompt ensembling with learned weights
- Learned class embeddings instead of raw text prompts
- Lightweight adapters as an alternative to full fine-tuning

---

5. Closing remarks

If this project taught me anything, it's that your model is only as good as your data and how you're actually measuring it.
Prompt-based segmentation is cool and super flexible, but it's very picky. Change a word in your prompt, mess up an annotation, or have a bug in your eval script, and suddenly everything's off.
Honestly, I spent as much time fixing data issues and evaluation code as I did training models. The final system isn't perfect, but at least I understand what it's doing and why.
