Yeah — solid start. Your PHP is mostly okay and follows the right flow (save upload → call Python → call detector → return JSON). I ran a careful read and made a list of things to improve plus a safer, more production-ready version you can copy-paste.

Main issues & suggestions

1.

AI % calculation — you use average confidence (simple mean). Better to **weight by tokens** so long segments count more (recommended).

2.
3.

Error handling — if the Python extractor fails or returns invalid JSON, you should catch that and return a clear error. Same for DeepSeek errors.

4.
5.

Validate uploaded file — check extension and max file size before moving.

6.
7.

Paths on Windows vs Linux — your `pythonPath` uses a Windows venv path. Make this configurable via `.env` and fallback to `python3` on non-Windows.

8.
9.

Clamp / normalize confidences — ensure confidences are numeric and between 0–1.

10.
11.

Persist download token mapping — `download_token` should map to the generated annotated file path (store mapping in a small JSON file or DB). Currently you generate a token but don't save the mapping.

12.
13.

Security / info leak — don't return internal file paths to the client. Good you didn't, but keep it in mind.

14.
15.

Timeouts — `set_time_limit(300)` might still be too small for big docs + API calls; consider async job pattern for heavy files.

16.
17.

CORS / Response headers — you call `Utils::allowCors()` so OK — ensure content-type and proper error codes are used (you already use Utils::error / jsonResponse).

18.

Below is a **refactored version** of your `analyze.php` with the above improvements:

-

Token-weighted AI% calculation.

-
-

File type + size validation.

-
-

Python path from `.env` with fallback.

-
-

Defensive checks around PythonBridge and DeepSeekClient.

-
-

Save `download_token` → file mapping (simple JSON file inside uploads dir).

-
-

Cleaner rounding and normalized confidences.

-

Copy this in place of your current file (adjust `.env` keys and paths if needed).

```
<?phprequire_once dirname(__DIR__) . '/src/Utils.php';require_once
dirname(__DIR__) . '/src/PythonBridge.php';require_once
dirname(__DIR__) . '/src/DeepSeekClient.php';
Utils::loadEnv(dirname(__DIR__) . '/.env');Utils::allowCors();
// Allow longer processing for larger documents/API
callsset_time_limit(600); // increased
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    Utils::error('Only POST is allowed', 405);
}
if (!isset($_FILES['file'])) {
    Utils::error('Missing file upload', 400);
}
$upload = $_FILES['file'];if (!isset($upload['tmp_name']) ||
$upload['error'] !== UPLOAD_ERR_OK) {
    Utils::error('File upload failed', 400, ['code' =>
$upload['error'] ?? null]);
}
// Basic validation: extension and size$allowedExt = ['pdf',
'docx'];$maxBytes = (int) (Utils::env('MAX_UPLOAD_MB', 20) * 1024 *
1024); // default 20 MB
$origName = $upload['name'];$ext = strtolower(pathinfo($origName,
PATHINFO_EXTENSION));if (!in_array($ext, $allowedExt)) {
    Utils::error('Unsupported file type. Only PDF and DOCX allowed',
400);
}if ($upload['size'] > $maxBytes) {
    Utils::error('File too large', 413);
}
$segmentTokens = isset($_POST['segment_tokens']) ? (int)
$_POST['segment_tokens'] : 80;$threshold =
isset($_POST['threshold']) ? (float) $_POST['threshold'] :
0.65;$threshold = max(0.0, min(1.0, $threshold));
$uploadsDir = dirname(__DIR__) .
'/uploads';Utils::ensureDir($uploadsDir);
// make safe and unique filename$safeName = preg_replace('/[^a-zA-Z0-
9._-]/', '_', basename($origName));$storedPath = $uploadsDir . '/' .
uniqid('upload_', true) . '_' . $safeName;
if (!move_uploaded_file($upload['tmp_name'], $storedPath)) {
    Utils::error('Failed to store uploaded file', 500);
}
// Python path from env with fallback$pythonPathEnv =
Utils::env('PYTHON_PATH', '');if ($pythonPathEnv &&
file_exists($pythonPathEnv)) {
    $pythonPath = $pythonPathEnv;
} else {
    // fallback (works on Linux with python3) - on Windows set
PYTHON_PATH in .env
    $pythonPath = Utils::isWindows() ? null : 'python3';
    if ($pythonPath === null) {
        Utils::error('Python path not configured for Windows. Set
PYTHON_PATH in .env', 500);
    }
}
// call python extractor (defensive)$python = new
PythonBridge($pythonPath);try {
    $extracted = $python->extract($storedPath, $segmentTokens);
} catch (Exception $e) {
    // clean up uploaded file if needed
    //unlink($storedPath);
    Utils::error('Python extractor failed: ' . $e->getMessage(), 500);
}
if (!is_array($extracted) || !isset($extracted['segments'])) {
    Utils::error('Extractor returned invalid data', 500, ['raw' =>
$extracted]);
}
$segments = $extracted['segments'] ?? [];$totalTokens =
$extracted['total_tokens'] ?? array_reduce($segments, function($c,
$s){ return $c + (int)($s['tokens'] ?? 0); }, 0);
if (count($segments) === 0) {
    Utils::error('No text extracted from document', 400);
}
// Score segments via DeepSeek (wrap errors)$detector = new
DeepSeekClient();try {
    // detector->scoreSegments expected to add 'confidence' key to
each segment
    $scoredSegments = $detector->scoreSegments($segments);
} catch (Exception $e) {
    Utils::error('Detection service failed: ' . $e->getMessage(),
502);
}
// Normalize and validate confidences; ensure tokens field
exists$scoredSegments = array_map(function($seg) {
    $seg['tokens'] = isset($seg['tokens']) ? (int) $seg['tokens'] :
max(1, str_word_count($seg['text'] ?? ''));
    $conf = isset($seg['confidence']) ? floatval($seg['confidence']) :
0.0;
    // clamp
    $seg['confidence'] = max(0.0, min(1.0, $conf));
    // round for nicer JSON
    $seg['confidence'] = round($seg['confidence'], 4);
    return $seg;
}, $scoredSegments);
// Compute weighted AI percent by tokens$totalTokensForWeight =
array_reduce($scoredSegments, function($carry, $s) { return $carry +
($s['tokens'] ?? 0); }, 0);if ($totalTokensForWeight <= 0)
$totalTokensForWeight = 1;
$weightedSum = array_reduce($scoredSegments, function($carry, $s) {
    return $carry + (($s['tokens'] ?? 0) * ($s['confidence'] ?? 0.0));
}, 0.0);
$aiPercent = round(100.0 * ($weightedSum / $totalTokensForWeight), 2);
// create tokens and job ids$jobId =
Utils::makeToken('job');$downloadToken = Utils::makeToken('dl');
// persist download mapping (simple JSON map file) so /download can
lookup$downloadMapFile = $uploadsDir .
'/download_map.json';$downloadTargetPath = null; // actual annotated
file path will be created later by annotator$map = [];if
(file_exists($downloadMapFile)) {
    $raw = @file_get_contents($downloadMapFile);
    $map = $raw ? json_decode($raw, true) : [];
}$map[$downloadToken] = [
    'original' => $storedPath,
    'annotated' => null, // to be filled once annotator runs
    'created_at' => time()
];
@file_put_contents($downloadMapFile, json_encode($map,
JSON_PRETTY_PRINT));
// mark is_ai by threshold$segmentsOut = array_map(function ($seg)
use ($threshold) {
    $seg['is_ai'] = (($seg['confidence'] ?? 0.0) >= $threshold);
    return $seg;
}, $scoredSegments);
// response$response = [
    'job_id' => $jobId,
    'ai_percent' => $aiPercent,
    'total_tokens' => $totalTokensForWeight,
    'num_segments' => count($segmentsOut),
    'threshold' => $threshold,
    'segments' => $segmentsOut,
    'download_token' => $downloadToken,
];
Utils::jsonResponse($response);
```

A few final notes for your repo

-

Add `PYTHON_PATH` to `.env` on Windows (e.g. `C:\xampp\php\php.exe` for php? no—point to `C:\path\to\python.exe`). On Linux use `python3` by default.

-
-

Implement `/download` to read `uploads/download_map.json`, check `annotated` field and serve annotated file or return 404 if not generated; optionally kick annotator to produce annotated file on-demand.

-
-

Consider adding a small background job / queue if you expect big PDFs — synchronous HTTP may time out.

-
-

Log errors (Utils::logger or error_log) — helps debugging when extractor or detector fail.

-
-

Add unit tests / sample files in `examples/` for reproducible testing.

-

If you want I can:

-

generate a matching improved `PythonBridge.php` or `DeepSeekClient.php` to match this flow,

-
-

or create a simple `/download.php` that looks up the `download_map.json` and streams annotated files.

-

Which one do you want next?